

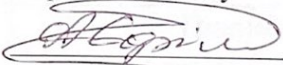
Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

ПОЯСНЮВАЛЬНА ЗАПИСКА
до випускної кваліфікаційної роботи бакалавра
на тему:
«Проект підсистеми захисту Web-ресурсів організації»

Виконав студент 4-го курсу, групи КБ-1
спеціальності 125 «Кібербезпека»

 Іванов І. В.

Керівник, завідувач кафедри КІ та КБ, к.т.н.

 Єфіменко А. А.

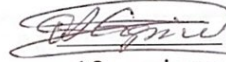
Рецензент, доцент кафедри КІ та КБ, к.т.н

 Пірог О. В.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 125 «КІБЕРБЕЗПЕКА»

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки

 Єфіменко А. А.
«12» квітня 2020 р.

ЗАВДАННЯ

на випускню кваліфікаційну роботу

Студента **Іванова Іллі Вікторовича**

Тема роботи: **Проект підсистеми захисту Web-ресурсів організації**

затверджена Наказом по університету від «12» квітня 2021 р. № 134/с

Термін здачі студентом закінченої роботи – **23 червня 2021 р.**

Вихідні дані для роботи:

1. Будова веб-додатків
2. Поширені вразливості веб-додатків
3. Розгортання вразливого додатку
4. Проведення атак, використання вразливостей
5. Рекомендації та техніки щодо захисту від наведених вразливостей

Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Єфіменко А. А.	12 квітня 2021	12 квітня 2021
2	Єфіменко А. А.	28 квітня 2021	28 квітня 2021
3	Єфіменко А. А.	12 травня 2021	12 травня 2021

Керівник



Єфіменко А. А.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка завдання	12 квітня 2021 – 13 квітня 2021	Виконано
2	Пошук, огляд та аналіз літературних джерел щодо наявних атак на веб-додатки	13 квітня 2021 – 20 квітня 2021	Виконано
3	Аналіз існуючих веб-технологій стосовно розробки	21 квітня 2021 – 23 квітня 2021	Виконано
4	Пошук інформації щодо існуючих веб-вразливостей	24 квітня 2021 – 26 квітня 2021	Виконано
5	Розробка загального плану виконання роботи	27 квітня 2021 – 28 квітня 2021	Виконано
6	Розгляд будови веб-додатків	30 квітня 2021 – 1 травня 2021	Виконано
7	Аналіз стеку вразливостей веб-додатків	2 травня 2021 – 4 травня 2021	Виконано
8	Аналіз конкретного списку веб-вразливостей відповідно до джерел	5 травня 2021 – 9 травня 2021	Виконано
9	Детальний опис та розбір окремих вразливостей	10 травня 2021 – 13 травня 2021	Виконано
10	Наведення рекомендацій щодо захисту від згаданих атак	14 травня 2021 – 18 травня 2021	Виконано
11	Розгортання вразливого додатку та його дослідження	19 травня 2021 – 22 травня 2021	Виконано
12	Пошук вразливостей у веб додатку та їх використання	23 травня 2021 – 24 травня 2021	Виконано
13	Проведення аналізу виконаних дій та наведення рекомендацій щодо захисту	25 травня 2021 – 29 травня 2021	Виконано
14	Підготовка презентації та виступу	30 травня 2021 – 31 травня 2021	Виконано
15	Отримання відгуку керівника та рецензії на кваліфікаційну роботу	13 червня 2021 – 20 червня 2021	Виконано
16	Захист	23 червня 2021	Виконано

Студент  Іванов І. В.

Керівник  Ефіменко А. А.

РЕФЕРАТ

Кваліфікаційна робота складається з теоретичних відомостей щодо будови веб-серверу, методів та засобів проведення веб-атак та захисту від них, моделі вразливого веб-додатку та, відповідно, моделювання атак та рекомендацій щодо захисту та пояснювальної записки. Пояснювальна записка до роботи містить 70 сторінок, 50 ілюстрацій та 1 таблицю.

Метою роботи є аналіз та розгляд існуючих веб-вразливостей та атак, а також пошук відповідних рішень щодо захисту систем.

В роботі проведено аналіз наявних вразливостей та атак на веб-додатки та наведено можливі варіанти рішення проблем. Також в роботі описано актуальність даної теми, та розказано про можливі наслідки від атак. Розгорнуто вразливий додаток та змодельовано атаки на нього, разом із наведенням рекомендацій щодо захисту. Наведено сценарій атаки та їх модельні приклади. Було також розглянуто основу будови веб-додатків та будову обраного вразливого веб-додатку. Наведено перелік аналогів вразливих додатків та веб-сайтів.

КЛЮЧОВІ СЛОВА: ЗАХИСТ, ВЕБ-ДОДАТОК, ВЕБ-САЙТ, АТАКА, OWASP, ВЕБ-ВРАЗЛИВІСТЬ, МОДЕЛЮВАННЯ.

КІК.КБ.Б-125-06-ПЗ				
Зм.	Арк.	№ докум.	Підпис	Дата
Розроб.		Іванов І. В.		22.06
Керівник		Єфіменко А. А.		22.06.21
Рецензент		Пірог О. В.		22.06.21
Зав. каф.		Єфіменко А. А.		22.06.21

Проект підсистеми захисту
Web-ресурсів організації

Пояснювальна записка

Літ.	Арк.	Аркушів
	4	70

Державний університет
«Житомирська
політехніка», група КБ-1

ABSTRACT

The qualification work consists of theoretical information on the structure of the web server, methods and means of conducting web attacks and protection against them, a model of a vulnerable web application and modeling attacks and recommendations regarding protection from the given attacks and explanatory note. The explanatory note to the work contains 70 pages, 50 illustrations and 1 table.

The purpose of this work is to analyze and review existing web vulnerabilities and attacks, as well as to find appropriate solutions to protect systems.

The work analyzes the existing vulnerabilities and attacks on web applications and presents possible solutions. The work also describes the relevance of this topic, and talks about the possible consequences of attacks. Deployed a vulnerable application and simulated attacks on it, along with providing recommendations for protection. The attack scenario and their model examples are given. The basis of the structure of web applications and the structure of the selected vulnerable web application were also considered. The list of analogues of vulnerable applications and websites was given.

KEY WORDS: DEFENSE, WEB APPLICATION, WEBSITE, ATTACK, OWASP, WEB VULNERABILITY, SIMULATION.

					КИК.КБ.Б-125-06-ПЗ	
						5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

XSS – Cross Site Scripting

ОС – операційна система

CSRF – Cross-site request forgery

OWASP – Open Web Application Security Project

WAF – Web application firewall

HTTP – HyperText Transfer Protocol

IP – Internet Protocol

XML – Extensible Markup Language

URL – Uniform Resource Locator

WASC – Web Application Security Consortium

CWE – Common Weakness Enumeration

ЗМІСТ

ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ ВЕБ-ДОДАТКІВ ТА ЇХ ВРАЗЛИВОСТЕЙ	10
1.1. ПРИНЦИПИ ПОБУДОВИ ВЕБ-ДОДАТКІВ ТА ОРГАНІЗАЦІЇ ВЕБ-РЕСУРСІВ	10
1.2. АНАЛІЗ ВРАЗЛИВОСТЕЙ ВЕБ-ДОДАТКІВ ТА ВЕБ-РЕСУРСІВ	12
1.3. АНАЛІЗ НАЙПОШИРЕНІШИХ ВРАЗЛИВОСТЕЙ ВЕБ-ДОДАТКІВ	15
РОЗДІЛ 2. АНАЛІЗ АТАК НА ВЕБ-РЕСУРСИ, МЕТОДИ ТА ТЕХНОЛОГІЇ ЇХ РЕАЛІЗАЦІЇ	21
2.1. АНАЛІЗ НАЙПОШИРЕНІШИХ АТАК НА ВЕБ-РЕСУРСИ	21
2.2. МЕТОДИ ТА ЗАСОБИ ПРОВЕДЕННЯ ВЕБ-АТАК	25
2.3. МЕТОДИ ТА ЗАСОБИ ЗАХИСТУ ВІД ВЕБ-АТАК	34
РОЗДІЛ 3. МОДЕЛЬНИЙ ПРИКЛАД ПРОВЕДЕННЯ АТАКИ ТА ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ВЕБ-ДОДАТКУ	45
3.1. ПРОЕКТ ІНФРАСТРУКТУРНОГО РІШЕННЯ ВРАЗЛИВОГО ВЕБ-ДОДАТКУ	45
3.2. РЕАЛІЗАЦІЯ АТАКИ ДЛЯ МОДЕЛЬНОГО ПРИКЛАДУ	50
3.3. РЕАЛІЗАЦІЯ ЗАСОБІВ ЗАХИСТУ ВІД АТАК НА ВЕБ-ДОДАТКИ	62
ВИСНОВКИ	67
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	68

ВСТУП

Актуальність теми. Кількість веб-сайтів та сервісів, які вони пропонують зростає кожного дня. Веб-технології використовуються всюди – чи то додаток який ви відкриваєте у браузері, чи то бот в улюбленному месенджері.

Веб-додатки пропонують зручне управління банківськими рахунками, оплату комунальних послуг, замовлення квитків у кіно, планування подорожей – але на ряду з стрімким розвитком технологій, які є корисними для людей, розвиваються і технолоґії, які можуть нашкодити. Хакери та їх здібності розвиваються так само стрімко, як і інша, корисна частина інтернету, але вони не афішують свій прогрес. Радше, здобуті знання вони використовують задля отримання доступу до конфіденційних та інших цінних даних, або доступу до окремих інтернет-сервісів, які відкривають їм безліч можливостей.

Веб-сайти зараз вимагають високої стабільності, тому що відповідають за дуже важливі процеси в різних сферах нашої життєдіяльності. Якщо вже не говорити про бізнес та те, які гроші там циркулюють, та які збитки несуть компанії за хвилини простою, то варто виділити державний сектор чи медицину.

Функціонування відповідних сервісів може бути набагато важливішим ніж заробіток грошей.

Важливість кібербезпеки в цілому і так зрозуміла – без неї, сучасному уявленню про Інтернет було б важко існувати. В даній роботі буде тісно розглянуто важливу частину безпеки Інтернету – безпеку веб-додатків.

Спеціалісти, які сфокусовані на безпеці веб-додатків, кожен день захищають користувацьку інформацію та чутливі дані загалом. Разом з цим мова йде і про вже згадану стабільність та працездатність бізнесу. А бізнес сьогодні – різний.

Актуальність обраної теми полягає в тому, що обізнаність в захисті веб-сервісів та додатків має зменшити, якщо не повністю нейтралізувати злочинний вплив в деяких випадках. Відповідні методи захисту дозволять підтримати працездатність додатків та безпеку внутрішньої інформації.

Метою кваліфікаційної роботи є аналіз та розгляд існуючих веб-вразливостей та атак, а також пошук відповідних рішень щодо захисту систем.

Встановлена мета обумовлює наступні завдання:

- аналіз найпоширеніших методик атак існуючих вразливостей;
- детальний перегляд деякої кількості вразливостей;
- розгляд та вибір засобів та технік захисту від наведених вразливостей;
- розгортання вразливого веб-додатку та його дослідження;
- моделювання раніше розглянутих атак та використання вразливостей у наближеному до реальності середовищі;
- рекомендації щодо захисту від практично змодельованих прикладів атак.

Об'єктом дослідження є процес атаки на веб-додатки та захист чи можливі поради щодо захисту від цих чи подібних атак та вразливостей.

Предметом дослідження є методи і засоби проведення атак та захисту від них.

В процесі виконання кваліфікаційної роботи було розгорнуто функціонуючий, але навмисно вразливий веб-додаток, знайдено відповідні вразливості, реалізовано їх використання у корисних цілях, де можливо; проаналізовано та наведено поради щодо уникнення таких атак.

РОЗДІЛ 1. АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ ВЕБ-ДОДАТКІВ ТА ЇХ ВРАЗЛИВОСТЕЙ

1.1. Принципи побудови веб-додатків та організації веб-ресурсів

Веб-додатки - це програми, які призначені для створення інтерфейсу між користувачем та веб-сервером. Веб-сервер – це комп'ютер, на якому працює програма веб-додатку та місце, де можуть зберігатися бази даних, файли та інші дані та ресурси користувачів або компаній.

Усі веб-додатки базуються на одній базовій архітектурі – «клієнт-сервер» та мають загальний протокол обміну даними – HTTP.

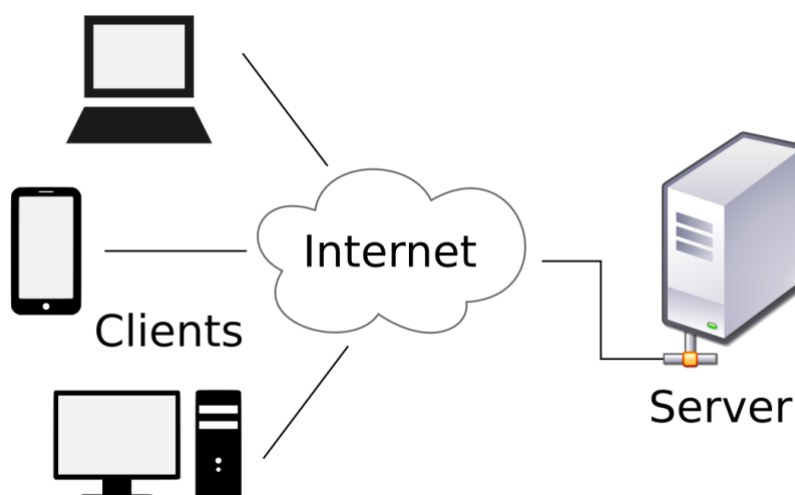


Рисунок 1.1 – Клієнт-серверна архітектура

Задля успішної роботи даної архітектури, виконуються наступні пункти:

1. Клієнт посилає HTTP запит на сервер
2. Сервер отримує запит, виконує відповідні дії та обчислення, формує відповідь та надсилає її користувачу (клієнту), також за допомогою протоколу HTTP.

«Клієнт» у даному контексті, це сторона, яка ініціює запит. Клієнтом може бути комп'ютер, програмне забезпечення, або і людина, яка використовує відповідне програмне забезпечення.

Сервер, в свою чергу, це програмне забезпечення яке обробляє запити та дає на них відповіді. Веб-сервер також може бути клієнтом – наприклад, він є клієнтом серверу баз даних чи просто іншого веб-серверу.

На стороні веб-сервера працює програмний додаток. Він може бути написаний будь-якою мовою, яка вміє працювати із протоколом HTTP. Метою такої програми є формування відповідей на запити клієнта. Відповіддю може бути текст, картинка, інший тип медіа файлів, відповідь у вигляді об'єкту та інше. Також деяка інформація може передаватися у заголовках запитів.

Взагалі, веб-сайти поділяються на статичні та динамічні. Статичні – це ті, які ніяк не інтерпретують запит на стороні сервера та статичні сторінки HTML видаються сервером в тому вигляді, у якому вони збережені: поза залежністю від змінних, що передаються із запитом. Предназначені, скоріше, для презентації. У даній роботі мова йтиметься про динамічні веб-сайти. Ті, які є інтерфейсом у постійному спілкуванні між клієнтом та сервером.

Динамічні веб-сайти, вочевидь, складні з точки зору технічної організації: вони постійно спілкуються із серверами баз даних, з іншими сервісами задля отримання нової інформації, та одночасно мають постійно надавати цю інформацію користувачам, як тільки вони отримають запит на це.

Зазвичай, для зручності керування контентом, є додаткова сторінка, якщо не ціла система – панель адміністратора або «система керування вмістом» - CMS, за допомогою яких можна змінювати наповнення веб-сторінок.

Найпоширенішою мовою програмування для веб-додатків є Node.js [22] - платформа з відкритим кодом для виконання високопродуктивних мережових застосунків, написаних мовою JavaScript. Стек технологій є дуже обширний, деякі популярні технології будуть зазначені на рисунку 1.2.

А в браузері можна виконувати лише JavaScript. Існує багато фреймворків - інфраструктури програмних рішень для полегшення розробки складних систем, але всі вони в кінці компілюються в звичайний JavaScript код та виконуються у браузері.

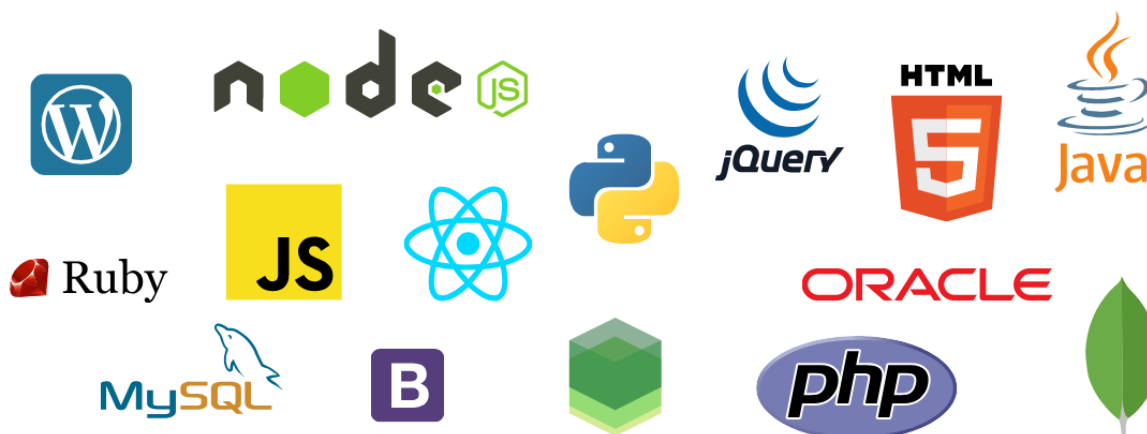


Рисунок 1.2 – Деякий перелік веб-технологій

1.2. Аналіз вразливостей веб-додатків та веб-ресурсів

Малий бізнес, банки та великі промислові концерни залежать від веб-додатків. Шукаючи товар чи послугу, ми також відвідуємо веб-сайт компанії. Завдяки розвитку веб-технологій державні послуги покращились як за доступністю, так і за якістю. Веб-сайти є публічним обличчям як бізнесу, так і уряду, тому будь-які проблеми з веб-сайтом можуть зашкодити репутації. З цієї та інших причин власники веб-додатків мотивовані вдосконалювати та підтримувати свої веб-сайти. Належна увага до кібербезпеки є ключовою частиною в цій сфері.

Веб-додатки в основній мірі є загальнодоступними ресурсами, тобто, знаючи URL адресу сайту, кожен може його відкрити. Через легку доступність та постійний розвиток мережі інтернет та його користувачів, веб-сайти стають вразливими до атак, або просто недоброчесних людей, які можуть намагатися погіршити якість роботи ресурсу, або внутрішні дані чи файли.

Під атакою мається на увазі порушення нормальної працездатності веб-серверу, модифікація, видалення або публікація, можливо й секретного вмісту.

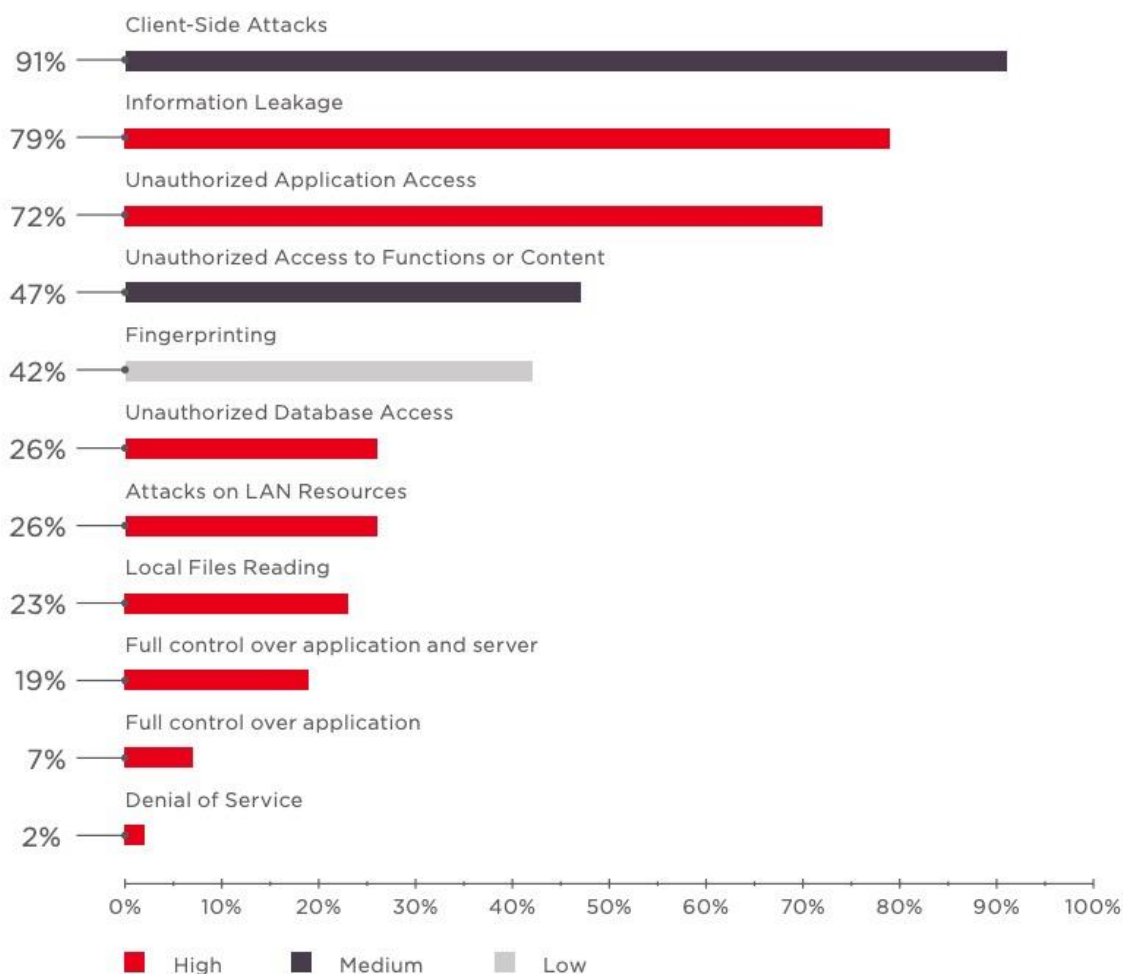


Рисунок 1.3 – Найпоширеніші загрози (відсоток перевірених веб-додатків)

Згідно з дослідженням порталу Positive Technologies [12], майже кожна веб-програма містить вразливості, які дозволяють атакувати користувачів. У більшості випадків винні міжсайтові сценарії (XSS). Цього року відсоток зріс ще більше. Лише одна така вразливість може спричинити серйозні наслідки, в чому можна впевнитись побачивши купу заголовків, кажучих про це в новинах.

Як видно на рисунку 1.3, саме атаки на клієнтській стороні є найбільш поширеними, тому можна зробити висновок, що найбільшою вразливістю веб-сайту є саме тей шар, з яким працюють користувачі, а особливо чутливими є місця, де можна вводити дані, які потім будуть відправлені на сервер.

Також на рисунку 1.4 показано стек вразливостей веб-додатків і саме взаємодія з додатками там теж знаходиться на першому місці.

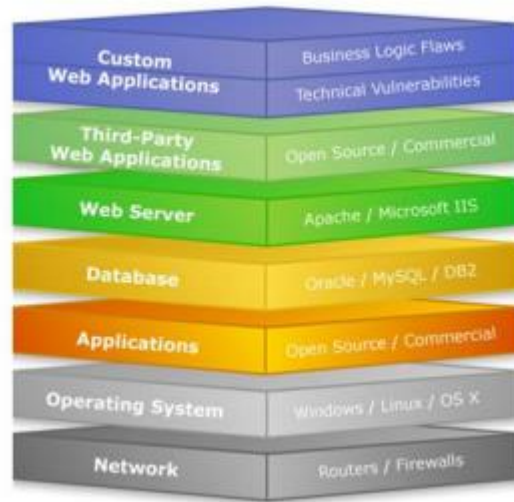


Рисунок 1.4 – Стек вразливостей веб-додатку

Це пов'язано з тим, які дані можуть бути відправлені на сервер. У звичайному сценарії роботи та користування ресурсом це буде просто збереження якоїсь інформації та її передача. У разі спроби виконання зловмисних цілей – відправляються дані, які містять дещо більше ніж просто користувацьку інформацію. Про це буде детально згадано у наступних розділах.

Такі вразливості можна віднести до погано написаного коду та «дір» у ньому, бо при належній фільтрації та обробці будь-яка інформація втратить властивість бути небезпечною.

З розвитком, та через те що складність веб-додатків постійно зростає, все більше розробників користуються вже готовими інструментами: шаблонами, програмними модулями та інш. Це вимагає додаткової дбалості при виборі таких інструментів, адже це є продукт розроблений іншими людьми і перед додаванням таких інструментів до своєї бібліотеки, потрібно провести ретельну перевірку коду на вразливості. Більшість розробників цим нехтують і в результаті, коли у такому готовому інструменті знаходять вразливість – всі сервіси, у яких присутній цей додаток автоматично знаходяться під загрозою і потребують швидкого реагування.

Наступними ступіннями можливих загроз є саме веб-сервер, операційні системи, системи управління базами даних і так далі. Це все також є програмним

продуктом, але ці компоненти не є підконтрольними розробнику веб-сайту і зазвичай вони достатньо захищені, через великий досвід та «вік» - багато вразливостей були виправлені у минулому.

Більш того, навіть використовуючи захищені та перевірені програми, додатки та бібліотеки, можна залишити вразливості просто неправильно налаштувавши ці елементи системи. Дуже важливо слідкувати за цими кроками, щоб не виставити дані користувачів на показ через неувімкнену захисну функцію.

Окрім цього, треба тримати все програмне забезпечення постійно оновленим.

1.3. Аналіз найпоширеніших вразливостей веб-додатків

Скануванням та аналізом або підтриманням переліку вразливостей займається деяка низка організацій, таких як: cgsecurity, WASC (Web Application Security Consortium) [23], CWE (Common Weakness Enumeration) [24], OWASP [21] та інші. Дана робота буде посилатися на інформацію з порталу OWASP.

Проект Open Web Application Security Project - це некомерційна фундація, яка працює над підвищенням безпеки програмного забезпечення. Фонд OWASP є джерелом інформації щодо захисту Інтернету для розробників та технологів зі всього світу, завдяки програмному забезпеченню з відкритим кодом під керівництвом спільноти, десяткам тисяч членів цієї спільноти та провідним освітнім та навчальним конференціям.

OWASP Top 10 [29] - це документ з інформацією про безпеку веб-додатків для розробників. Він представляє широкий консенсус щодо найбільш критичних небезпек та ризиків у веб-додатках.

Згідно цього документу, основні та найпоширеніші атаки та вразливості наведені далі:

- ін'єкції, такі як SQL, NoSQL, OS та LDAP
- порушена автентифікація
- розголошення, або відображення чутливих даних

- XML External Entities (XXE)
- порушений контроль доступу
- неправильні налаштування безпеки
- Cross-Site Scripting (XSS)
- десеріалізація
- використання компонентів з відомими вразливостями
- недостатність логів та моніторингу

Ін'єкції, такі як SQL, NoSQL, OS та LDAP виникають коли ненадійні дані надсилаються інтерпретатору як частина команди або запиту. Дані, які вводить зловмисник можуть змусити програму виконати ненавмисні дії або отримати доступ до даних без належного дозволу. Це виникає внаслідок класичної невдалої фільтрації введення даних. Наприклад: форма введення коментарів.

Порушена автентифікація: Функції веб-додатків пов'язані з автентифікацією та управлінням сесіями часто реалізуються неправильно, що дозволяє зловмисникам компрометувати паролі, ключі або токени сесій, або використовувати інші вразливості, щоб тимчасово або назавжди прийняти ідентифікаційні дані інших користувачів.

Розголошення, або відображення чутливих даних: Багато веб-додатків та API не захищають конфіденційні дані належним чином. Не важливо, фінансові, охоронні чи ідентифікаційні дані, зловмисники можуть викрасти або модифікувати таку слабо захищену інформацію для здійснення шахрайства, наприклад з кредитними картками, викраденням особистих даних або інших злочинів. Без додаткового захисту, конфіденційні дані можуть бути скомпрометовані навіть в стані спокою або під час їх передачі, тому і вимагають особливих заходів обережності при обміні ними з браузером.

XML External Entities (XXE): Багато старих або погано налаштованих обробників коду XML сприймають та оброблюють зовнішні сутності в документах XML. Ці зовнішні сутності можуть використовуватися для розкриття

внутрішніх файлів, наприклад за допомогою обробника URI, сканування портів, віддаленого виконання коду та атак типу відмови в обслуговуванні.

Порушений контроль доступу: Обмеження щодо того, що користувачі можуть чи не можуть робити, часто не застосовуються належним чином. Зловмисники можуть використовувати ці недоліки для несанкціонованого доступу до функцій та / або даних, таких як доступ до облікових записів інших користувачів, перегляд конфіденційних файлів, зміна даних інших користувачів, зміна прав доступу тощо.

Неправильні налаштування безпеки: Неправильні налаштування безпеки є дуже поширеною проблемою. Зазвичай це результат конфігурацій за замовчуванням; неповних або спеціальних конфігурацій, відкритого хмарного сховища, неправильно налаштованих заголовків HTTP та детальних повідомлень про помилки, що містять конфіденційну інформацію. Всі операційні системи, фреймворки, бібліотеки та програми повинні бути не лише надійно налаштовані, але й вчасно оновлені.

Cross-Site Scripting (XSS): XSS атаки трапляються, коли програма відображає або опрацьовує ненадійні дані на новій веб-сторінці, без попередньої належної перевірки чи екранування, або оновлює вже завантажену веб-сторінку неперевіреними даними, наданими користувачем, за допомогою API браузера, який може створювати нові HTML або JavaScript елементи. XSS дозволяє зловмисникам виконувати деякі сценарії в браузері жертви, які можуть викрадати сесии користувачів, псувати веб-сайти або перенаправляти користувача на інші, шкідливі сайти.

Десеріалізація. Небезпечна десеріалізація часто призводить до віддаленого виконання коду. Навіть якщо атаки десеріалізації не призводять до віддаленого виконання коду, їх можна використовувати для виконання деяких атак згаданих раніше, включаючи атаки повторного відтворення, ін'єкції та атаки на зміну привілеїв.

Використання компонентів з відомими вразливостями: Компоненти, такі як: бібліотеки, фреймворки та інші програмні модулі, працюють з тими

					КІК.КБ.Б-125-06-ПЗ	
						17

самими привілеями, що і сама програма. Якщо використовується вразливий компонент, така атака може сприяти серйозній втраті даних або захопленню сервера. Програми та API, що використовують компоненти з відомими вразливостями можуть підірвати захист додатків та активізувати різноманітні атаки.

Недостатність логів та моніторингу: Недостатність логів та моніторингу в поєднанні з відсутністю або неефективною інтеграцією з реагуванням на інциденти, дозволяє зловмисникам продовжувати атакувати системи, підтримувати стійкість, переходити до більшої кількості систем і продовжувати підробляти, витягувати або знищувати дані. Більшість досліджень показують, що час виявлення порушення становить понад 200 днів, тим більше загрози виявляються зовнішніми сторонами, а не внутрішніми процесами чи моніторингом.

Вище зазначені лише декілька широко поширених вразливостей – насправді їх набагато більше. До цього списку можна додати ще безліч атак та вразливостей. Розглянемо ще кілька прикладів.

Ще одна поширена та важлива річ у безпеці додатку: **Відсутній контроль доступу на рівні функції.**

Насправді, це просто помилка авторизації. Це означає, що при виклику функції на сервері не виконувалась належна авторизація. Розробники часто покладаються на той факт, що сервер згенерував інтерфейс і користувачі не можуть отримати доступ до функцій, які їм не показані. Це не так просто, оскільки зловмисник може знайти “приховану” функціональність та запити, незважаючи, що цієї функціональності немає в інтерфейсі. Наприклад, кнопка яка відкриває адмін-сторінку доступна лише користувачу, який насправді є адміністратором. Ця кнопка перенаправляє користувача на посилання «/admin». При відсутній авторизації, зловмиснику достатньо просто перейти по цьому посиланню і він отримає доступ до адмін-панелі, замість того щоб пройти перевірку на наявність доступу.

Розглянемо випадок підробки міжсайтових запитів (CSRF).

					КІК.КБ.Б-125-06-ПЗ	
						18

Це гарний приклад атаки, коли зловмисна програма обманює браузер, використовуючи свої повноваження. Наприклад, сторонній сайт може змусити браузер користувача зробити щось для зловмисника. Так цей сторонній сайт надає запит цільовому сайту (наприклад, банку), використовуючи ваш браузер із вашими файлами cookie / сеансом. Наприклад, якщо ви ввійшли у ваш банківський акаунт на одній вкладці, і ця сторінка є вразливою до атаки, запити з іншої вкладки можуть змусити ваш веб-браузер робити зловмисні дії під вашими даними.

Зловмисник Джон хоче перевести собі кошти з гаманця Ненсі. Банк Ненсі вразливий до CSRF атаки. Щоб надіслати гроші, Джон повинен виконати такий запит:

<http://bank.com/transferFunds?value=1000&destination=123456789>

Після відкриття такої сторінки, виконується успішний переказ коштів на зазначений рахунок. Зловмисник, тобто Джон, знає, що Ненсі користується сайтом, підконтрольний йому. Для виконання атаки, Джону достатньо розмістити елемент, наприклад, зображення, яке замість справжньої картинки буде містити запит на посилання, зазначене вище. Далі, коли Ненсі відкриє зловмисний веб-сайт, її браузер автоматично спробує завантажити зображення, по заданому посиланню. Перейшовши по посиланню, Ненсі посилає запит своєму банку на переказ коштів до акаунту Джона.

У наступному розділі буде наведена більш детальна інформація щодо деяких атак та їхнього виконання.

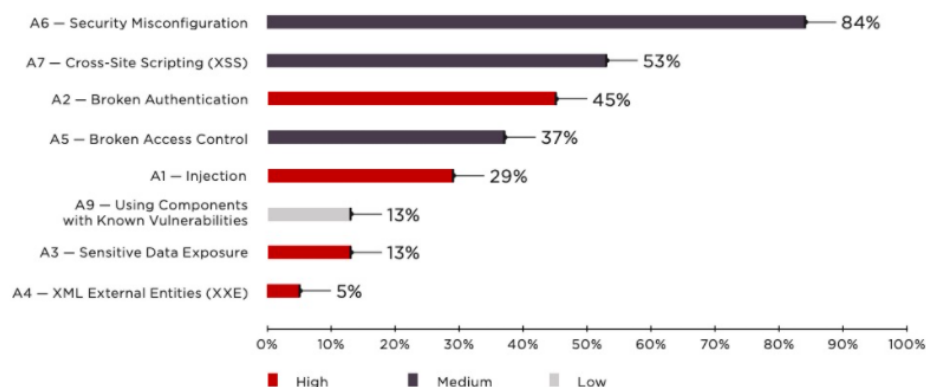


Рисунок 1.5 – Статистика вразливостей веб-додатків

На рисунку 1.5 та таблиці 1.1 показана статистика вразливостей веб-додатків – автоматичне сканування.

Таблиця 1.1

Статистика вразливостей веб-додатків (автоматичне сканування)

Тип уразливості	% Вразливостей	% Вразливих сайтів
Cross-Site Scripting	30,08	50,10
Information Leakage	29,82	97,19
Insufficient Transport Layer Protection	11,18	53,25
Fingerprinting	9,59	45,68
SQL Injection	7,95	15,50
Malware detect	5,52	1,32
Improper Parsing	3,92	6,62
HTTP Response Splitting	0,84	2,07
SSI Injection	0,42	0,42
Remote File Inclusion	0,22	0,44
Path Traversal	0,23	0,70
OS Commanding	0,08	0,06
Content Spoofing	0,06	0,19
Denial of Service	0,05	0,23
Insufficient Authorization	0,02	0,10
Brute Force	0,01	0,06
Directory Indexing	0,01	0,05

РОЗДІЛ 2. АНАЛІЗ АТАК НА ВЕБ-РЕСУРСИ, МЕТОДИ ТА ТЕХНОЛОГІЇ ЇХ РЕАЛІЗАЦІЇ

2.1. Аналіз найпоширеніших атак на веб-ресурси

Згідно проаналізованих даних у попередньому розділі було встановлено, що найбільш вразливим місцем кожного додатку є його користувачі, тому варто приділяти більше уваги на все, що пов'язане з користувачами, їх діями та інформацією.

Здебільшого, на кожному веб-сайті є форми для відправки даних на сервер: коментарі, повідомлення, заповнення будь-яких інших форм. Такий функціонал дає можливість зловмисникам відправляти ін'єкції на сервер. Є декілька видів ін'єкцій. Розглянемо тип атаки, спрямований саме на сервер та його внутрішні дані та складові.

Окрім користувацьких даних, компрометованими можуть бути й інші джерела: змінні середовища, зовнішні та внутрішні веб-служби.

Цей вид вразливості дуже поширений, особливо у застарілому коді. Ін'єкції часто зустрічаються в запитах SQL, LDAP, XPath або NoSQL, командах ОС, обробниках XML, заголовках SMTP та запитах ORM.

Недоліки легко виявити при вивченні коду, а автоматизовані сканери та «розмивачі» можуть допомогти зловмисникам їх знайти.

Додаток вразливий до атак, коли:

- Дані, надані користувачем, не перевіряються, не фільтруються та не оброблюються додатком.
- Динамічні запити або непараметризовані виклики без контекстного екранування використовуються безпосередньо в інтерпретаторі коду.
- Зловмисні дані використовуються в параметрах пошуку об'єктно-реляційного відображення (ORM) для вилучення додаткових, конфіденційних записів.

Деякі з найпоширеніших ін'єкцій були згадані раніше. Концепція однакова серед них усіх. Перегляд вихідного коду є найкращим методом виявлення, чи додатки вразливі до ін'єкцій. Слід ретельно проводити автоматизоване тестування всіх параметрів, заголовків, URL-адрес, файлів cookie, даних JSON, SOAP та XML. Для виявлення ін'єкцій, організації можуть використовувати статичні (SAST) та динамічні (DAST) інструменти тестування додатків.

Ін'єкція SQL - це вразливість, яка дозволяє зловмиснику втручатися в запити, які додаток надсилає до своєї бази даних. Як правило, це дозволяє отримати та переглядати дані, які зазвичай не є доступними. Це може включати інформацію, яка належить іншим користувачам, або решту даних, до яких має доступ сама програма. У багатьох випадках зловмисник спробує змінити або видалити ці дані, спричиняючи постійні зміни у вмісті або поведінці програми.

Зловмисник може спробувати посилити цю атаку для компрометації сервера або іншої внутрішньої інфраструктури, або просто виконати атаку відмови в обслуговуванні (DoS).

Успішна атака такого типу може призвести до несанкціонованого доступу до приватних, конфіденційних даних: паролі, дані кредитної картки або особиста інформація. Багато гучних резонансів за останні роки були результатом атак ін'єкцій. Вони призвели до погіршення репутації компаній та навіть штрафів. Іноді, зловмисник може отримати постійний бэкдор (доступ) до систем організації, що може призвести до довгострокового втручання і може залишатися непоміченим протягом тривалого періоду. Вплив на бізнес залежить від того, наскільки залежним він є від самого додатку та даних які він надає.

Іншим, але схожим видом атаки є **Cross-site scripting**. Атаки Cross-site scripting (також відомі як XSS) - це вразливість, яка дозволяє зловмисникові контролювати взаємодію користувачів із вразливим веб-додатком. Це дозволяє зловмисникові обходити політику «same origin», яка призначена для розмежування, відокремлення веб-сайтів один від одного. XSS вразливості зазвичай дозволяють зловмиснику маскуватися під користувача-жертву,

виконувати будь-які дії від їх лица, до яких у користувача є доступ, та переглядати та контролювати доступні дані. Якщо користувач-жертва має привілейований доступ, зловмисник може отримати повний контроль над даними та функціями додатка.

Цей вид атаки також вважається ін'єкцією, але різниця між цим видом та попередньо розглянутим є у тому, що зловмисник має намір працювати з користувачем, а не проводить атаку напряму на сервер.

Cross-site scripting маніпулює вразливим додатком так, щоб той повертав користувачам шкідливий JavaScript. Коли такий код виконується всередині браузера жертви, зловмисник може повністю контролювати їх взаємодію з додатком.

Існує три основних типи XSS-атак:

- Reflected XSS
- Stored XSS
- DOM-based XSS

У випадку **Reflected XSS**, шкідливий код походить з поточного HTTP запиту. У **Stored XSS**, шкідливий сценарій походить із бази даних веб-сайту. **DOM-based XSS** - вразливість знаходиться в коді на стороні клієнта, а не в коді на стороні сервера. Більш детально зазначені атаки будуть розглянуті у наступних пунктах.

Атака **CSRF** була згадана у попередньому розділі. Її основна мета – виконувати дії від лица жертви, але спосіб реалізації відрізняється від атаки XSS.

Щоб атака CSRF стала можливою, мають бути дотримані три ключові умови:

- у додатку має бути функціонал, яку зловмисник має підставу викликати/виконати. У цьому випадку, задача – саме виконати дію. Дія може бути привілейована (наприклад, зміна дозволів для інших користувачів) або будь-яка інша конкретна дія з даними користувача (наприклад, зміна власного пароля).

– обробка запитів відбувається на основі файлів cookie. Виконання дії передбачає виконання одного або декількох HTTP запитів, а програма, для ідентифікації користувача, покладається виключно на файли cookie. Іншого механізму для відстеження сеансів або перевірки запитів не має існувати.

– не повинно бути непередбачуваних параметрів в запиті. Запити, що виконують дію, не мають містити жодних параметрів, значення яких зломисник не може визначити або вгадати. Наприклад, при зміні паролю, функція не є вразливою, якщо зломиснику потрібно знати значення вже існуючого пароля.

Припустимо, що програма має функцію, яка дозволяє користувачу змінити пароль від власного акаунту. При виконанні такої дії, користувач робить запит HTTP, як показано нижче:

POST /email/change HTTP/1.1

Host: example-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 30

Cookie: sessionId= KajSIyFshVd4I10wwXkKkajSIyFshVd4I10wwXkK

Body: [email=example@gmail.com](mailto:example@gmail.com)

Такий запит відповідає умовам, необхідним для виконання атаки CSRF:

– дія (зміна електронної адреси) цікава зломиснику. Після цього, зломисник може спробувати скинути пароль задля отримання повного доступу до акаунту.

– для ідентифікації користувача, додаток використовує лише значення cookie. Інших механізмів для відстеження сеансів користувачів не існує.

– зломиснику не треба визначати значення додаткових параметрів запиту, необхідних для виконання дії.

Якщо жертва знаходитиметься на підконтрольному зломиснику сайті, зломиснику достатньо буде викликати якусь функцію по типу відправки форми. Ця форма буде перенаправляти дані на вразливий веб-сайт, та буде містити інформацію, вигідну для злодія. При виконанні цієї дії, браузер жертви виконає HTTP запит. Якщо користувач зареєстрований до вразливої сторінки, браузер

автоматично підставить значення сесії в cookie. Так, вразливий веб-сайт обробить запит у звичайному режимі - розгляне його як запит, зроблений користувачем-жертвою та змінить його електронну адресу.

У чому різниця між XSS та CSRF атаками? Міжсайтовий сценарій (XSS) дозволяє зловмиснику виконувати будь-який JavaScript код у браузері жертви. Підробка міжсайтових запитів (CSRF), в свою чергу, дозволяє зловмиснику «змушувати» жертву виконувати деякі дії.

Наслідки вразливостей XSS, як правило, більш серйозні, ніж для вразливостей CSRF:

CSRF застосовується лише до деяких дій, які може виконати жертва. Багато додатків все ж таки реалізують захист від CSRF, але можуть пропустити чи ігнорувати одну чи дві дії, які в результаті опиняються викритими. Успішний експлойт XSS, навпаки, може змусити користувача виконати **будь-які** дії, до яких у користувача є доступ, незалежно від того, в яких функції виникає вразливість.

CSRF можна також охарактеризувати як «односторонню» вразливість, так як зловмисник може змусити жертву виконати якусь дію, але він не зможе отримати відповідь чи результат виконання цієї дії. XSS, в свою чергу, є «двостороннім», тому що скрипт, який виконує зловмисник, може робити довільні запити та, що важливо, отримувати та читати відповіді від цих запитів.

2.2. Методи та засоби проведення веб-атак

Існує широкий спектр вразливостей, атак та методів SQL ін'єкцій, які виникають у різних ситуаціях. Деякі загальні призначення атаки включають в себе:

- Отримання прихованих даних
- Зміна логіки програми
- UNION атаки
- Вивчення структури бази даних
- Сліпа SQL ін'єкція

У випадку отримання прихованих даних, можна змінити SQL запит, щоб той повернув деякі додаткові дані.

Зміна логіки програми - запит змінюється так, що він перешкоджає звичайній логіці програми.

UNION атаки - отримання даних з різних таблиць бази даних.

Вивчення структури бази даних - отримання інформації про версію та структуру бази даних.

Сліпа SQL ін'єкція - після відправки запиту, зловмиснику невідомо результат роботи цієї дії.

Розглянемо атаку типу «отримання прихованих даних».

Є деякий веб-додаток, який відображає інформацію по відповідним запитам. Наприклад, інтернет магазин, який сортує товари по категоріям. Для того, щоб отримати відсортований список товарів, користувач натискає на відповідну кнопку, яка в свою чергу виконує наступний запит:

[https:// website.com/showProducts?category=games&type=1](https://website.com/showProducts?category=games&type=1)

Програма виконає наступний запит до бази даних:

*SELECT * FROM products WHERE category = ' games' AND type = '1' and accessible = '1'*

Що має означати, що програма запрошує у бази даних наступну інформацію:

- отримати всі деталі (*)
- з таблиці «products»
- де категорія - games
- і продукт має бути доступним (=1)

Параметр «*accessible = 1*» присутній для того, щоб база даних повернули лише ті продукти, які мають бути доступними для продажу або перегляду. В іншій ситуації, цей параметр дорівнюватиме 0, що означатиме, що продукти не мають бути показаними звичайним користувачам.

Якщо в додатку немає жодних захисних механізмів від атак SQL ін'єкцій, то злодій може використати наступний запит:

					КІК.КБ.Б-125-06-ПЗ	
						26

<https://website.com/showProducts?category=games&type=1'-->

Веб-сервер, в свою чергу, інтерпретує цей запит у наступному вигляді:

SELECT * FROM products WHERE category = ' games' AND type = '1'--' and accessible = '1'

Ключовим елементом такої дії є те, що послідовність подвійних тире - це коментар в мові SQL, і означає, що все, що знаходиться після цих символів – вважається коментарем та не сприймається в самому запиті. Таким способом, злоумисник позбувся усіх додаткових перевірок у цьому запиті, а саме, в даному прикладі, продукти не будуть відфільтровані по критерію доступності і база даних поверне усі продукти з даної вибірки, навіть якщо товар не має бути показаний та доступний користувачу.

Далі, ворожий запит можна вдосконалити ще більше, запросивши програму відобразити всі продукти з будь-яких категорій, навіть з тих, про які злоумисник не знає:

<https://website.com/showProducts?category=games'+OR+1=1-->

Інтерпретований запит буде виглядати так:

SELECT * FROM products WHERE category = 'games' OR 1=1-- and accessible = '1'

Такий запит поверне всі продукти де категорія = ігри, або 1=1. Так як вираз «1=1» завжди є вірним а інші перевірки зроблено закоментованими, то результатом виконання цього запиту буде повернення абсолютно всіх продуктів, в будь-якій категорії, з будь-якими захисними флагами.

Зміна логіки програми:

Зазвичай, вхід на веб-сайти реалізований просто за допомогою введення логіна та пароля. Коли користувач входить за допомогою свого логіна user та пароля password, програма робить наступний SQL-запит, який поверне всю інформацію про знайденого користувача:

SELECT * FROM users WHERE username = 'user' AND password = 'password'

Вхід у систему буде вважатися успішним якщо запит поверне результат.

Інакше, це означає що користувача з такими даними не знайдено.

					КІК.КБ.Б-125-06-ПЗ	
						27

Так, при відсутньому захисті, зломисник може використати вже згаданий раніше метод з використанням коментарів у запиті. Нехай логін користувача буде «admin'--», а пароль – будь-яке значення, або навіть пусте поле.

Запит буде виглядати так:

```
SELECT * FROM users WHERE username = 'admin'--' AND password = 'does not matter'
```

В результаті, такий запит поверне всю інформацію про користувача з логіном “admin”, бо пароль і інші дані навіть не будуть перевірені.

Отримання інформації з інших таблиць бази даних (**UNION атака**):

У випадках, коли додаток повертає результати запиту на сторінку до користувача, можна виконати UNION атак, яка виконає додатковий, вказаний запит та поверне більше інформації ніж функція повинна була.

Додамо до оригінального запиту конструкцію UNION, як показано нище, та отримаємо не тільки інформацію про продукти, але й дані користувачів з зовсім іншої таблиці.

```
SELECT itemName, itemPrice FROM products WHERE category = 'games'  
UNION SELECT username, password FROM users--'
```

Якщо зломиснику відомо, що додаток вразливий до інєкцій, перед тим як проводити попередньо згадані атаки, слід отримати більше інформації про базу даних та її структуру.

Можна навіть виконати команду, яка поверне повний перелік існуючих таблиць. Формат запиту залежить від бази даних яка використовується, але у більшості з них, наступний запит спрацює:

```
SELECT * FROM information_schema.tables
```

Такий запит можна зробити за допомогою конструкції UNION, розглянутої вище.

Багато випадків реалізації SQL ін'єкцій - це сліпі ін'єкції. В такому випадку, програма не повертає результати запиту або деталі помилок у своїх

відповідях. Цю вразливість все ще можна використовувати для доступу до несанкціонованих даних, але це, як правило, є більш складним процесом.

Зловмисник може змінити логіку запиту, щоб отримати помітну різницю у відповіді програми. Як приклад, можна включити введення якоїсь нової умови логічну операцію:

Веб-додаток перевіряє належність значень cookie до якогось користувача наступним запитом:

```
SELECT userId FROM users WHERE cookieId = 'KajSIyFshVd4I10wwXkK'
```

При успішній відповідності, програма не поверне ніяких результатів пошуку, але, наприклад, перенаправить користувача на іншу сторінку, що буде означати успішність виконання дії та запиту.

Зловмисник скомпрометує значення cookie та перетворить його у додаткову умову для запиту:

```
SELECT userId FROM users WHERE cookieId = 'KajSIyFshVd4I10wwXkK' and password = 'brute'--
```

Таким чином, можна робити перевірку другого параметру. У нашому випадку – пароль. Якщо веб-додаток у результаті зробить перенаправлення на іншу сторінку, це означатиме успішність виконання запиту. Обидва параметри ї вірними – пароль вгадано. Можна застосувати атаку грубої сили задля перебору значень паролю, але слід використовувати складніші запити для кращого аналізу.

XSS атаки:

Reflected cross-site scripting (Reflected XSS) виникає коли додаток отримує якісь дані через HTTP запит та одразу використовує ці дані на сторінці без попередніх обробок.

Припустимо, що існує фільтр пошуку, який доступний за даною адресою:

https:// website.com/search?q=games

Веб-додаток, отримуючи такий запит, починає пошук, повертає дані та відображає на сторінці, що саме запросив користувач. Відображає таким способом:

<p>Знайдео 10 товарів за запитом: games</p>

					КІК.КБ.Б-125-06-ПЗ	29

У разі, якщо параметр запиту не був оброблений захисними функціями, атаку можна реалізувати вже за допомогою трохи зміненого вмісту запита:

`https:// website.com/search?q=<script>/*Attacker's code*/</script>`

Так, на веб-сайті буде відображена відповідь, що товарів за даною категорією не знайдено, а вразливий код буде виконаний на стороні клієнта.

Якщо зловмисник може керувати скриптом, який виконується у браузері жертви, тоді він скоріше за все, зможе повністю скомпрометувати цього користувача. Зловмисник може:

- виконати будь-яку дію на веб-сайті, яку може виконати користувач
- переглянути будь-яку інформацію, яку може переглядати користувач
- змінити будь-яку інформацію, яку може змінити користувач

Існують різні способи, за допомогою яких зловмисник може змусити жертву зробити відповідний запит для здійснення атаки. Це може бути розміщене посилання на веб-сайті, яким керує зловмисник, або на іншому веб-сайті, де можна генерувати вміст, або посилання, надіслане в електронному листі, твіті чи іншому повідомленні. Атака може бути націлена безпосередньо на окремого користувача або може здійснити атаку на велику кількість будь-яких користувачів веб-сайту.

Варіантів реалізації такої атаки може бути багато, а перевірка чи обробка даних зі сторони веб-додатку впливає на те, які дані мають передаватися.

Stored XSS виникає, коли додаток отримує дані з ненадійного джерела, а потім відображає ці дані на своїх веб-сторінках без належного попереднього оброблення.

Компрометовані дані можуть бути збережені будь-яким способом – для приклада, нехай це буде залишеним коментарем до фільма. При цьому, такі дані можна навіть вписати в поле «Ім'я», або «Псевдонім», а не в тексті самого коментаря. Дані можуть надходити і з інших джерел, наприклад, додаток електронної-пошти, що показує повідомлення, отримані через SMTP; додаток, що відображає повідомлення з соціальних мереж та інше.

Простим та зрозумілим прикладом реалізації даної атаки розглянемо наступний приклад:

Зловмисник відправляє просте повідомлення в чат, із «поганим» вмістом – замість звичайного тексту, він відправить деякі html теги чи одразу скрипти, які призведуть до виконання зловмісного коду або його завантаження з інших ресурсів, та подальшого виконання.

У випадку, якщо повідомлення не обробляється, усі користувачі даного чату отримують повідомлення та їх браузери виконують скрипт.

Ключова відмінність між Reflected та Stored XSS полягає в тому, що Stored XSS атаки містяться в самому додатку. Зловмиснику не потрібно знаходити спосіб заманити жертву, щоб та перешла по шкідливому посиланню. Зловмисник просто розміщує скрипт у самому додатку і чекає, поки інші користувачі натраплять на нього.

Також, використання Reflected XSS актуальне лише у ситуаціях, коли вразливість стосується користувачів, які залогінені до веб-сайту. Атака у такому випадку частково полягається на вдачу, бо якщо запит буде зроблений у той момент, коли користувач не залогінений до системи, нічого не спрацює.

У випадку атаки Stored XSS, користувач стикається з ворожим кодом тільки коли він вже залогінений до сайту.

DOM XSS виникає, коли додаток JavaScript код на стороні клієнта обробляє дані з ненадійного джерела небезпечним способом, як правило – записує дані назад у DOM.

Якщо зловмисник контролює поле введення, він може легко відправити шкідливе значення, яке буде одразу виконане після виконання функції.

Приклад шкідливого значення:

Результати пошуку: `<img src=lorem onerror='/* Шкідливий код */>`

Додаток приймає код, зазначений вище у полі пошуку і відображає його на сторінці. Схожий спосіб працює у Reflected XSS атаці, але тут замість виконання коду напряду, зловмисник намагається додати картинку на сторінку, але він

вказав невірний шлях до неї. Це значить, що коли додаток не знаходить у своїх файлах такої картинки, спрацьовує функція onerror, яка містить шкідливий код.

Розглянемо проблеми та типові помилки з авторизацією. Авторизація - це застосування обмежень щодо доступу до ресурсів для окремого користувача або функції. У веб-додатках, у процесі автентифікації основу роль грає сеансовий ключ (токен), який зберігається та передається у cookies.

1. Спочатку користувач проходить автентифікацію – додаток ідентифікує користувача та підтверджує, що він є тим, ким себе називає.

2. Далі, за допомогою сесій, додаток знає який користувач робить які подальші запити – щоб позбутися повторної процедури автентифікації.

3. Функції контролю доступу, в свою чергу, при кожному запиті, визначають чи користувач має достатній рівень привілеїв щоб виконати дію або отримати інформацію по відповідному запиту.

Дуже поширеною проблемою є невиконання 3-го пункту, описаного вище – реалізація функцій контролю доступу, що вважається критичною вразливістю. Розробка та управління даною частиною є складною та динамічною проблемою, яка має враховувати дуже багато факторів, які потім мають бути технічно впроваджені. А враховуючи те, що проектування відбувається не автоматично, а людьми, то існує високий потенціал помилок.

З точки зору користувача, функції контролю доступу поділяються на 3 категорії:

Вертикальний контроль доступу. З таким способом реалізації, різні користувачі мають доступ до різних функцій та ресурсів додатка. Адміністратор може створювати та видаляти акаунти, а звичайний користувач – ні.

Горизонтальний контроль доступу – це коли користувачі мають однаковий набір привілеїв, але використовуючи одні й ті ж функції – вони отримують лише свої набори даних і не мають доступу до чужих. Приклад: коли користувачі переглядають свій профіль, вони всі користуються однаковими функціями, але інформацію отримують лише відповідно до свого акаунту та його потреб – вони не можуть переглянути інформацію іншого користувача, бо не мають доступу.

					КІК.КБ.Б-125-06-ПЗ	
						32

Контроль доступу, який залежить від контексту – додаток забороняє виконання деяких функцій, залежно від деякої логіки. Наприклад, користувач вже не може змінювати окреме замовлення після того як воно було сплачене.

Загалом, якщо користувач може отримати доступ до функції або даних, до якої він не мав би мати доступу – це проблема **Broken Access**, описана вище.

Така вразливість виникає коли «чутливим» функціям не приділяється належна увага, в плані їх захисту. Логіка проблеми проста, але її наявність може призвести до дуже поганих наслідків та великих збитків.

Наприклад, існує деякий функціонал або спеціальні сторінки, які мають бути доступними лише для адміністраторів. Звичайно, цей функціонал не показано користувачу – кнопки та посилання не відображені на сторінках, які переглядає звичайний користувач. Але лише це – не є належним захистом. Користувач може дуже просто вгадати або виявити вразливі та сховані сторінки і перейти до них не використовуючи для цього відповідних кнопок – ввівши адресу напряду, таку як: <https://example.com/admin>. Не маючи додаткового захисту, будь-хто може отримати доступ до адмін-панелі таким шляхом.

Вразливі сторінки навіть можуть бути згруповані у файлі robots.txt. Robots.txt – це протокол винятків для роботів, а саме, стандартний спосіб комунікації веб-сайтів з пошуковими та іншими роботами. Стандарт визначає, як повідомити веб-робота про те, які частини веб-сайту не повинні бути оброблені або відскановані.

Помилкою у горизонтальному контролі доступу є ситуація, коли користувач може отримати доступ до інформації, яка насправді належить іншому користувачу або не має бути доступна нікому. Скористатися такою вразливістю можна змінивши деякі параметри у наступному запиті: <https://example.com/profile?id=12>. У даному випадку можна спробувати передати id іншого користувача, та у випадку, якщо механізмів безпеки немає, додаток просто поверне чужу інформацію.

2.3. Методи та засоби захисту від веб-атак

Знаючи методи реалізації атак та використання вразливостей, можемо розглянути можливі методи та засоби захисту.

Методи захисту нічим не обмежені, так само як і варіації та кількість різноманітних атак. Після ідентифікації вразливості можна скористатися вже згаданими раніше ресурсами задля забезпечення належної ізоляції від подібних випадків у майбутньому.

Слід також розуміти, що одна і та сама атака може мати велику кількість варіантів виконання, тому обов'язково треба розглядати і такі потенційні вразливості. Знаючи вектор, треба час від часу оновлювати наявні технології захисту.

Розглянемо можливі варіанти захисту від атаки ін'єкцій. Основним вектором таких атак є дані, які додаток отримує від користувача. Найкращим підходом є контроль та перевірка введених даних та відстеження шаблонів атак. Уникнути таких атак допоможе виконання основних методів, описаних далі.

Перевірка вхідних даних. Цей процес спрямований на перевірку того, чи співпадає очікуваний зразок даних з насправді отриманим. Така перевірка гарантує, що формат даних, їх довжина чи тип відповідають очікуваному вводу.

Якщо приймається текстове значення, де користувач має заповнити власну інформацію, чи щось подібне, варто перевіряти введені значення регулярними виразами. Таким чином, можна буде легко перевірити, чи справді користувач ввів свою e-mail адресу, мобільний телефон, ім'я чи поштовий індекс.

У разі, коли потрібно обрати значення із фіксованого набору (випадаючий список, перемикач) – розробник точно знає, що він очікує отримати і може легко перевірити чи передана інформація є вірною. Наприклад, якщо треба обрати своє місто з випадаючого списку, дуже легко перевірити наявність введеного зразка з справжнім списком міст і викрити поганий запит.

```
function doesCityExist(city) {
  switch (city) {
    case "Житомир":
      return true;
    case "Київ":
      return true;
    case "Львів":
      return true;
    case "Одеса":
      return true;
    default:
      return false;
  }
}

doesCityExist( city: "Вінниця"); // false
```

Рисунок 2.1 – Приклад функції для перевірки введених значень

На рисунку 2.1 показано приклад, як можна обробити дані, прийняті з користувацької сторінки. У даному випадку, це жорстка перевірка, бо користувач має обрати тільки таке значення, перелік яких йому запропоновано.

Дані, отримані зовні повинні бути завжди перевірені. Це правило поширюється не лише на дані, що надаються простими користувачами веб-додатку, але й на постачальників, партнерів, та інші сторони. Треті сторони, які начебто є довіреними, можуть бути також атаковані та зловмисні дані можуть бути передані і без їх відома.

Параметризовані запити - це засіб попередньої компіляції SQL запита. Цей метод дозволяє базі даних розпізнавати ворожий код та відрізнити його від справжніх вхідних даних.

Дані, які вводить користувач, автоматично «цитуються», і так, вони не спричинять нічого поганого. Цей стиль кодування допомагає пом'якшити атаку SQL ін'єкцій.

Параметризувати дані можна за допомогою вже розроблених інструментів, які є загально доступними. Наприклад, для Node.js можна використати модуль

Knex.js. Приклад роботи, а саме, параметризація запитів буде показана на рисунках нижче.

```
knex('users').where('id', 1)

Outputs:
select * from `users` where `id` = 1
```

Рисунок 2.2 – Параметризація запитів за допомогою модуля Knex.js

```
knex('users')
  .where((builder) =>
    builder.whereIn('id', [1, 11, 15]).whereNotIn('id', [17, 19])
  )
  .andWhere(function() {
    this.where('id', '>', 10)
  })

Outputs:
select * from `users` where (`id` in (1, 11, 15) and `id` not in (17, 19)) and (`id` > 10)
```

Рисунок 2.3 – Параметризація запитів за допомогою модуля Knex.js

Складність запитів може бути різною і приклади використання дуже обширно описані у документації цього модуля.

Збережені процедури – це один або декілька SQL запитів згруповані в логічну одиницю, яка виконується згідно вказаного плану. Подальші виклики такої процедури автоматично параметризують дані. Кожного разу замість написання нового запиту, можна буде використати таку збережену процедуру.

```
CREATE PROCEDURE GetProductDesc_withparameters
(@PID INT)
AS
BEGIN
SET NOCOUNT ON

SELECT P.ProductID,P.ProductName,PD.ProductDescription FROM
Product P
INNER JOIN ProductDescription PD ON P.ProductID=PD.ProductID
WHERE P.ProductID=@PID

END
```

```
EXEC GetProductDesc_withparameters 706
```

Рисунок 2.4 – Приклад створення та використання збереженої процедури

До збереженої процедури зображеної на рисунку 2.4 слід також додати параметризацію.

Escaping (екранування). В додачу до інших захисних технік, потрібно завжди використовувати функції екранування символів для даних, отриманих зовні. Ці функції надаються системами управління базами даних і використовуються для того, щоб СУБД ніколи не плутала введені дані з командою SQL. Наприклад, функція `mysql_real_escape_string()` у наступному PHP коді допоможе уникнути такої ситуації у сценарії:

```
<?php
$db_connection = mysqli_connect("localhost", "user", "password", "db");
$username = mysqli_real_escape_string($db_connection, $_POST['username']);
$password = mysqli_real_escape_string($db_connection, $_POST['password']);
$query = "SELECT * FROM users WHERE username = '" . $username . "' AND password = '" . $password . "'";
?>
```

Рисунок 2.5 – Приклад використання функції `mysql_real_escape_string`

Без використання цієї функції, код є вразливим до символу екранування «\» перед одинарними лапками. Наявність цієї невеликої зміни захистить від зловмисного запиту.

```
username: \
password: or 1 #

$query = select * from users where username=''.$username.' ' and password=''.$password.' ' ;

select * from users where username='\ ' or password=' or 1 # ' ;
```

Рисунок 2.6 – Приклад вразливого коду

На рисунку 2.6 показано як виглядає вразливий код та наведено приклад того, як виглядає зловмисний запит, який не був належно оброблений додатком.

Варто слідкувати за **привілеями акаунтів**, які працюють з базами даних.

Додаток не має використовувати облікові записи з адміністративним або просто високим рівнем доступу. Це можна робити лише за крайньої необхідності, оскільки таким чином зловмисники можуть отримати доступ до всієї системи. Це особливо небезпечно, якщо сервер баз даних використовується одразу кількома додатками та базами даних.

З цієї причини, щоб захистити програму від SQL ін'єкцій, краще захистити базу даних, використовуючи акаунт з найнижчим привілеєм. Кожна програма

має мати власні облікові записи для бази даних і ці облікові записи мають мінімальні права, необхідні додатку.

Замість того, щоб намагатись визначити, які права доступу слід забрати, краще зосередитись на визначенні того, які права доступу потрібні для коректної роботи, а все інше заборонити.

Такі методи, як перевірка вхідних даних, параметризовані запити, збережені процедури та екранування, добре працюють з різними векторами атак. Однак, через велику варіативність SQL-ін'єкцій, лише цих методів може бути недостатньо і окрім цього слід розглянути використання Web Application Firewall.

Web Application Firewall (WAF), або «брандмауер для веб додатків» - захищає веб-сервери на рівні додатків. У той час як традиційні брандмауери розділяють та фільтрують внутрішній та зовнішній мережевий трафік, WAF контролюють середу яка знаходиться між користувачами та веб-додатками.

Скануючи трафік, який передається по протоколу HTTP, WAF можуть ідентифікувати та захищати веб-додатки від таких атак як: SQL ін'єкції, міжсайтові сценарії XSS та відмова в обслуговуванні.

XSS атаки. Фактичні збитки цих атак залежать від природи додатку, його можливостей та збережених даних, а також від привілеїв компрометованого користувача.

У додатку, де всі користувачі анонімні, а вся інформація є публічною, користі від атаки буде мало. У випадку, якщо атака проводиться на ресурс, де зберігаються конфіденційні дані, банківська інформація, електронні листи чи медичні дані, вплив та збитки будуть, відповідно, серйозними. Особливо, якщо скомпрометовано буде користувача із більшим набором повноважень та доступу – в такому разі є ризик втратити контроль над додатком в цілому, разом зі всіма важливими даними.

Переважну більшість вразливостей XSS можна знайти за допомогою веб-сканерів вразливостей.

Тестування на Reflected та Stored XSS вручну, включає перевірку за допомогою введення деяких простих текстових значень у кожную точку вводу на веб-сайті, що допоможе визначити місце, куди введені дані повертаються та де відображаються. Потім визначається чи ці точки вводу є вразливі до атак XSS.

Ручне тестування атаки XSS на основі DOM, яка виконується при передачі даних в URL параметрах передбачає подібний процес: виконання простого унікального введення даних і подальше використання інструментів розробника браузера для пошуку DOM для цього введення та наступне тестування кожного такого місця, для визначення, чи є воно вразливим. Однак інші типи DOM XSS виявити важче. Щоб знайти вразливості на основі DOM у вхідних даних, що не передаються в URL-адресах (наприклад, `document.cookie`), або в помилках, що не засновані на HTML (наприклад, `setTimeout`), може зайняти дуже багато часу.

Політика безпеки вмісту (**CSP**) - це механізм браузера, спрямований на пом'якшення впливу міжсайтових сценаріїв та деяких інших вразливостей. Якщо додаток, який використовує CSP, має поведінку, яка схожа на XSS атаку, тоді CSP може перешкоджати або навіть запобігати виконанню вразливості.

Запобігання міжсайтовим сценаріям в деяких випадках є тривіальним - простим, але в інших випадках, це може бути набагато складнішим, залежно від складності додатку та того, як програма оброблює дані, які надходять від користувачів.

Ефективні методи щодо запобігання XSS вразливостей, передбачені у наступних пунктах:

- фільтрація вводу на вході
- кодування (екранування) даних на виході
- використання відповідних заголовків у відповідях
- політика безпеки вмісту

Фільтрація вводу на вході: у точці, де приймається введення даних користувачем, виконуйте якомога суворішу фільтрацію, виходячи з того, які дані очікуються від цієї точки вводу.

Використання відповідних заголовків у відповідях: щоб запобігти XSS атакам у HTTP відповідях, які не призначені для міщення HTML або JavaScript, слід використовувати заголовки типу «Content-Type» та «X-Content-Type-Options», задля впевненості, що браузері інтерпретують відповіді так, як заплановано.

Політика безпеки вмісту: в якості останньої лінії захисту можна використовувати політику безпеки вмісту (CSP), щоб зменшити ступінь небезпеки будь-яких уразливостей XSS, які все ще трапляються.

Кодування (екранування) даних на виході: коли введені користувачем дані виводяться у HTTP відповіді, вихідним даним слід бути закодованими, щоб запобігти їх інтерпретації як активний вміст (код). Залежно від контексту виводу, може знадобитися застосування комбінацій HTML, URL, JavaScript та CSS кодування.

Екранування слід застосовувати безпосередньо перед тим, як дані будуть виведені на сторінку. Переданий контекст може визначати яке саме кодування слід використати. Значення всередині рядка JavaScript можуть вимагати іншого типу екранування в порівнянні з тими, що знаходяться в контексті HTML.

Приклад екранування символів:

- " замінюється на "
- & замінюється на &
- < замінюється на <
- > замінюється на >

Екранування – це безперечно важливий елемент захисту від XSS, але лише цього недостатньо для повного запобігання вразливості в усіх випадках. Дуже важливо обробляти дані і при отриманні їх від користувача.

Розглянемо ситуацію: користувач заповнює інформацію про себе, де серед інших даних, має зазначити посилання на свій блог. Розробник точно знає який тип даних має отримати в цьому чи іншому полі вводу і тому, при отриманні

даних, має обов’язково перевірити чи співпадає тип даних з очікуваним, чи дані підходять по шаблону та інше.

У разі, якщо розробник не перевірить, що було уведено в поле, де мало б знаходитись посилання на свій блог, злоумисник може передати конструкцію, яка буде містити шкідливий протокол “javascript:”. Наприклад, значенням атрибута href у полі «Мій блог» буде «javascript:alert». При натисканні на посилання, користувач виконає злоумисний код.

Поля вводу можна також обмежувати за їх розміром.

При перевірці вхідних даних, ідеально - блокувати недійсне або неочікуване введення. В іншому випадку можна спробувати очистити або виправити такі погані дані. Але такий спосіб є більш схильним до помилок і його слід уникати, використовуючи перший метод – блокування такого вводу.

З приводу білих та чорних списків: бажано перевіряти вхідні дані по білим спискам, а не чорним. Замість того, щоб слідкувати та оновлювати списки шкідливих зразків даних чи протоколів, краще дозволяти лише ті дані, які співпадають з білим списком. Так, навіть якщо виникне новий спосіб маскування злоумисних даних, ми будемо дозволяти лише перевірені набори значень, символів, протоколів та шаблонів.

Щоб екранувати дані, введені користувачем, перед виведеннями HTML коду за допомогою JavaScript, можна створити власну функцію для базових цілей. Приклад такого коду наведено на рисунку 2.7 та результат його виконання на рисунку 2.8.

```
function escapeHTML(string){  
    return String(string).replace( searchValue: /[^\w. ]/gi,  replacer: function(c :string ){  
        return '&#'+c.charCodeAt(0)+'';  
    });  
}
```

Рисунок 2.7 – Функція для екранування HTML коду

CONSOLE

```
&#60;div id&#61;&#34;interface&#45;wrapper&#34;&#62;&#10;&#60;div id&#61;&#34;loading&#45;screen&#34;&#62;&#10;&#60;div class&#61;&#34;loading&#45;screen&#45;container&#34;&#62;&#10;&#60;div class&#61;&#34;loading&#45;screen&#45;image&#34;&#62;&#60;&#47;div&#62;&#10;&#60;div class&#61;&#34;loading&#45;screen&#45;message&#34;&#62;How much&#63;&#60;&#47;div&#62;&#10;&#60;&#47;div&#62;&#10;&#60;&#47;div&#62;&#10;&#60;div id&#61;&#34;mute&#45;modal&#34; class&#61;&#34;modal modal&#45;fixed&#45;footer&#34;&#62;&#10;&#60;div class&#61;&#34;mute&#45;progress progress&#34;&#62;&#10;&#60;div class&#61;&#34;pbar indeterminate&#34;&#62;&#60;&#47;div&#62;&#10;&#60;&#47;div&#62;&#10;&#60;div class&#61;&#34;modal&#45;content&#34;&#62;&#10;&#60;div id&#61;&#34;mute&#45;modal&#45;center&#34;&#62;
```

На рисунку 2.9 буде зображено символи, на які символи реагує вищезазначений код.

Таку функцію доречно використовувати у всіх випадках, коли треба відобразити дані, введені користувачем. Приклад використання:

Параметр “innerHTML” може інтерпретувати передані дані як код. Подібний параметр “textContent” виводить дані як текст, тому є безпечнішим, але різні функції використовуються для різних цілей, тому не можуть бути замінені.

Схожим способом можна екранувати значення, які будуть використовуватися у коді JavaScript. Приклад коду та інші деталі будуть наведені на рисунках нижче.

```
1 function escapeJS(string){
2   return String(string).replace(/([^\w. ])/gi, function(c){
3     return '\\u'+('0000'+c.charCodeAt(0).toString(16)).slice(-4);
4   });
5 }
6
7
8 console.log(escapeJS(`eval('console.log("process.env.secret_token")')`));
```

Рисунок 2.9 – Функції для екранування JS вставок

```
"Running fiddle"
"eval\u0028\u0027console.log\u0028\u0022process.env.secret_token\u0022\u0027\u0029"
```

Рисунок 2.10 – Результат виконання функції для JS вставок

Така функція забезпечить захист від ворожих даних, коли потрібно використати інформацію, наприклад, у змінній.

Політика безпеки вмісту - це остання лінія захисту від міжсайтових сценаріїв. Якщо запобігання XSS не вдається навіть при дотриманні всіх вищезазначених технік, CSP допоможе пом'якшити атаку, обмеживши можливості зловмисника.

CSP дозволяє контролювати різні речі, наприклад, чи можна завантажувати зовнішні скрипти та чи будуть виконуватися внутрішні сценарії. Для увімкнення CSP потрібно додати HTTP заголовок, який називається Content-Security-Policy, з відповідним значенням.

Приклад такого заголовку:

default-src 'self'; script-src 'self'; object-src 'none'; frame-src 'none'; base-uri 'none';

Налаштування, зазначені вище, визначає, що такі ресурси як зображення та скрипти, можна завантажувати лише з того самого джерела, що і головна сторінка. Отже, навіть якщо зловмисник може успішно виконати атаку XSS, він зможе завантажувати ресурси лише з відповідних джерел (у даному випадку лише з поточного джерела). Це сильно перешкоджає та знижує ймовірність того, що зловмисник зможе успішно використати вразливість та нашкодити.

CSRF. Простий та надійний спосіб від CSRF атак – додати відповідний токен до запитів, які ідуть на сервер. Такий токен має бути:

- непередбачуваним та згенерований випадковим чином – містити букви та цифри різного регістру, можливо й деякі символи
- індивідуальним для кожного користувача, прив'язаним до сеансу

– обов'язково перевірятися перед виконанням будь-якої дії від лиця користувача.

Деякі веб-сайти захищаються від CSRF атак за допомогою атрибуту SameSite для cookie.

Атрибут SameSite слід використовувати для контролю над тим, як файли cookie подаються у міжсайтових запитах. Встановивши атрибут для cookie, які містять сеансовий токен, програма може запобігти стандартній поведінці браузера, який додає файли cookie до запитів, незалежно від того, звідки вони походять. Атрибут SameSite додається до Set-Cookie, коли сервер видає файл cookie і цьому атрибуту можна надати два значення: Strict або Lax.

Якщо значення атрибута SameSite встановлено як "Strict" - браузер не включатиме cookie в запити, що надходять з інших сайтів. Це безпечно, але погіршує взаємодію з користувачем - якщо зареєстрований користувач переходить за посиланням з іншого сайту – він не буде автоматично залогінений і йому доведеться увійти на сайт знову.

При встановленому значенні «Lax», браузер включатиме cookie у запити з інших сайтів лише якщо запит використовує метод GET і запит походить з навігації найвищого рівня та дія була ініційована користувачем (натискання на посилання). Інші запити будуть проігноровані.

РОЗДІЛ 3. МОДЕЛЬНИЙ ПРИКЛАД ПРОВЕДЕННЯ АТАКИ ТА ЗАБЕЗПЕЧЕННЯ ЗАХИСТУ ВЕБ-ДОДАТКУ

3.1. Проект інфраструктурного рішення вразливого веб-додатку

Для моделювання атак було вирішено використовувати вже вразливий додаток, розгорнутий на локальній машині. В інтернеті наявний як найменш десяток таких додатків. Перелік деяких з них наступний:

- Buggy Web Application (BWAPP)
- CTFlearn
- Damn Vulnerable IOS App (DVIA)
- Damn Vulnerable Web Application (DVWA)
- Defend the Web
- Google Gruyere
- Hack.me
- HackTheBox
- HackThisSite
- Hellbound Hackers
- ThisIsLegal – Are You?
- Game of Hacks

Серед цих додатків безкоштовні та платні варіанти. Кожна з них має свій спосіб демонстрації, перевірки та виконання тих чи інших дій. Але у нашому випадку, для виконання атак було обрано OWASP Juice Shop – це безкоштовний додаток з відкритим програмним кодом та великою кількістю вразливостей.

Будучи веб-додатком з великою кількістю передбачуваних вразливостей, OWASP Juice Shop є протилежністю best practices або стандартів та шаблонів для веб-розробників: це інструмент для обізнаності, навчання, демонстрації та вправ з вразливостями в сучасних веб-додатках. OWASP Juice Shop [26] - це некомерційний проект із відкритим кодом, заснований організацією OWASP, який розробляється та підтримується волонтерами.

Juice Shop містить 100 завдань різної складності, де можна спробувати використати основні вразливості. Ці вразливості були навмисно закладені в програму саме з метою навчання, але таким чином, як би воно могло трапитись і в реальних проектах!

Juice Shop - це веб-додаток, реалізований на JavaScript та TypeScript. На фронт-енді, для створення односторінкового додатку, використовується фреймворк Angular.

JavaScript також використовується на серверній частині і є єдиною мовою програмування. Додаток на базі Express працюючий на Node.js, доставляє клієнтський код у браузер. Він також надає необхідний функціональний набір для клієнта через RESTful API. В якості базової бази даних був обраний легкий SQLite через його файлову структуру.

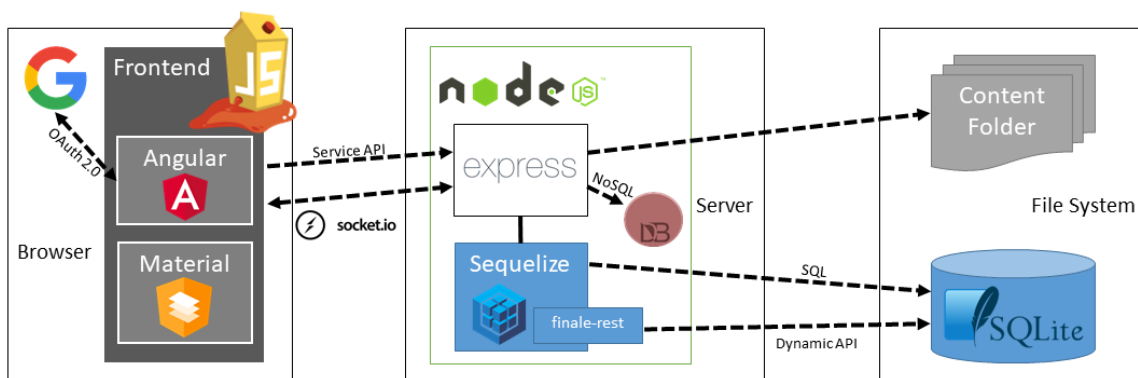


Рисунок 3.1 – Діаграма зв'язків компонентів додатку

На схемі, зображеній на рисунку 3.1, показані шляхи зв'язку компонентів додатку між клієнтом, сервером та даними.

Вразливості, які можна знайти в OWASP Juice Shop, поділяються на декілька різних класів. Більшість із них охоплюють різні типи вразливостей з відомих списків або документів, таких як OWASP Top 10, OWASP ASVS, OWASP Automated Threat Handbook [27], OWASP API Security Top 10 та OWASP Top 10 Risks Privacy Privacy або MITRE's Common Weakness Enumeration. На діаграмі зображеній на рисунку 3.2, представлено «розподілення» загроз, ризиків та атак, серед усіх вразливостей веб-додатку.

Challenges Category Distribution

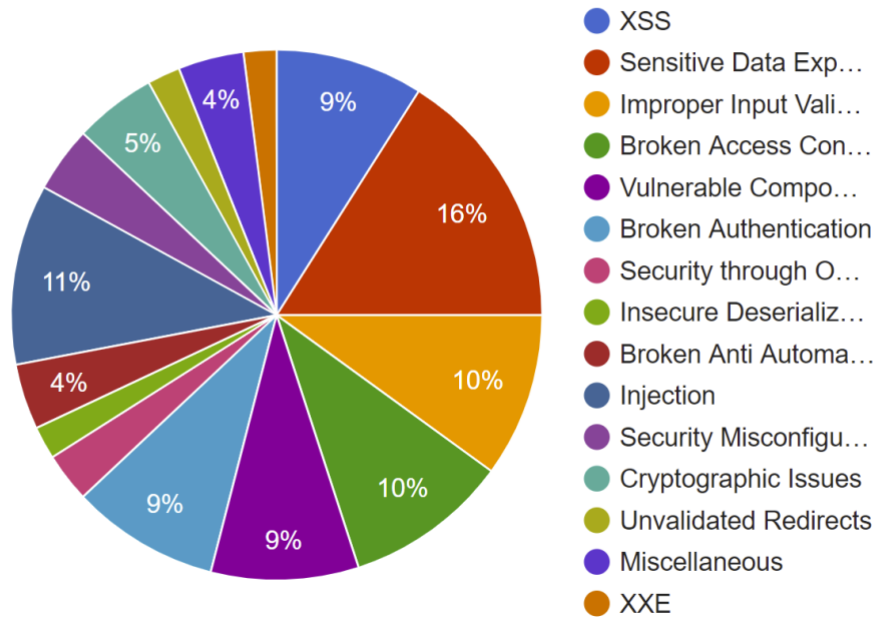


Рисунок 3.2 – Діаграма розподілення вразливостей у додатку

Вихідний код проекту можна завантажити з відповідного GitHub репозиторію.

Після клонування репозиторію на локальну машину, необхідно встановити додаткові модулі, для повноцінної роботи додатка.

```
test/server/verifySpec.ts(303,9): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(305,32): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(305,42): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(305,52): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(312,9): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(314,32): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(314,42): error TS2532: Object is possibly 'undefined'.
test/server/verifySpec.ts(314,52): error TS2532: Object is possibly 'undefined'.

added 1950 packages, and audited 1951 packages in 6m

112 packages are looking for funding
  run 'npm fund' for details

13 vulnerabilities (3 moderate, 7 high, 3 critical)

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.

Run 'npm audit' for details.
```

Рисунок 3.3 – Процес завантаження додаткових модулів

На рисунку 3.3 можна побачити, що для коректної роботи додатку було завантажено 1950 додаткових модулів. Декілька основних, високорівневих модулів можуть посилатися на кілька низькорівневих модулів, а ті на ще нижчі і так далі – тому виходить так багато залежностей. Загалом, для розгортання додатку може знадобитись до 1.5ГБ вільної пам'яті.

Размер: 1,13 ГБ (1 218 128 416 байт)
 На диске: 1,25 ГБ (1 349 095 424 байт)
 Содержит: Файлов: 95 144; папок: 14 424

Рисунок 3.4 – Розмір розгорнутого додатку

```
$ npm start
> juice-shop@12.8.0 start
> node build/app

info: All dependencies in ./package.json are satisfied (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Detected Node.js version v14.16.1 (OK)
info: Detected OS win32 (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Required file server.js is present (OK)
info: Required file main-es2018.js is present (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Required file tutorial-es2018.js is present (OK)
info: Required file polyfills-es2018.js is present (OK)
info: Required file runtime-es2018.js is present (OK)
info: Required file vendor-es2018.js is present (OK)
info: Required file main-es5.js is present (OK)
info: Required file tutorial-es5.js is present (OK)
info: Required file polyfills-es5.js is present (OK)
info: Required file runtime-es5.js is present (OK)
info: Required file vendor-es5.js is present (OK)
info: Port 3000 is available (OK)
info: Server listening on port 3000
```

Рисунок 3.5 – Веб-сайт запущено

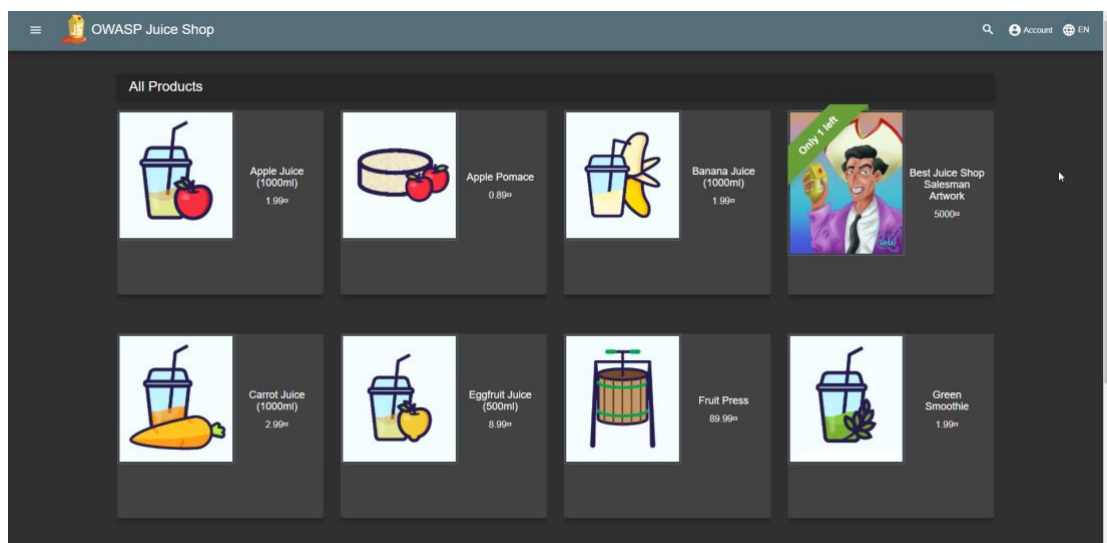


Рисунок 3.6 – Вигляд веб-сторінки

На рисунках 3.5 та 3.6 відповідно, показано як додаток було запущено із командного рядка та вигляд працюючого веб-сайту.

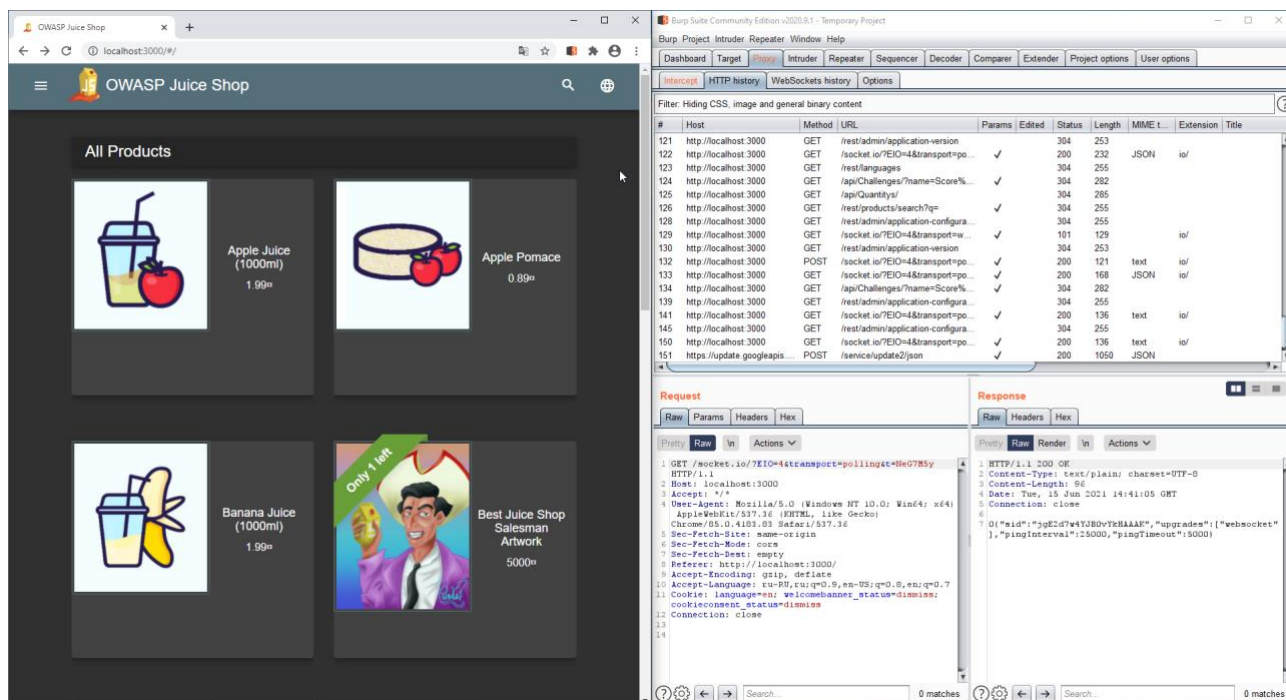


Рисунок 3.7 – Перехоплення запитів через додаток Burp Suite

Запити, через які додаток отримує та відправляє інформацію на сервер можна аналізувати і за допомогою стандартних інструментів розробника у сучасному веб-браузері. Але для зручності використання було вирішено обрати програму Burp Suite.

Burp або Burp Suite [25] - це набір інструментів, що використовуються для тестування веб-додатків на проникнення. Він розроблений компанією під назвою Portswigger, яка також є псевдонімом її засновника Dafydd Stuttard. BurpSuite прагне бути універсальним набором інструментів і його і так широкі можливості можна розширити, встановивши доповнення - VApps.

Це найпопулярніший інструмент серед професійних дослідників безпеки веб-додатків та мисливців за помилками («баг-хантери»). Простота використання дає перевагу серед безкоштовних альтернатив, таких як OWASP ZAP [28].

3.2. Реалізація атаки для модельного прикладу

Вразливий додаток, який ми будемо розглядати – це інтернет магазин. Як і в кожному інтернет магазині, тут є стандартний набір функцій: пошук товарів, їх відображення, відгуки; реєстрація, власний кабінет. Кожна з цих функцій потенційно вразлива.

Першим чином, переглядаючи товари в інтернет магазинах, ми користуємось фільтрами та пошуком. У попередніх розділах було розглянуто ситуацію, що коли веб-сайт показує на сторінці параметри пошуку, можна спробувати використати вразливість Reflected XSS, де задані зловмисні дані передаються через HTTP запит, а жертва переходить по такому посиланню.

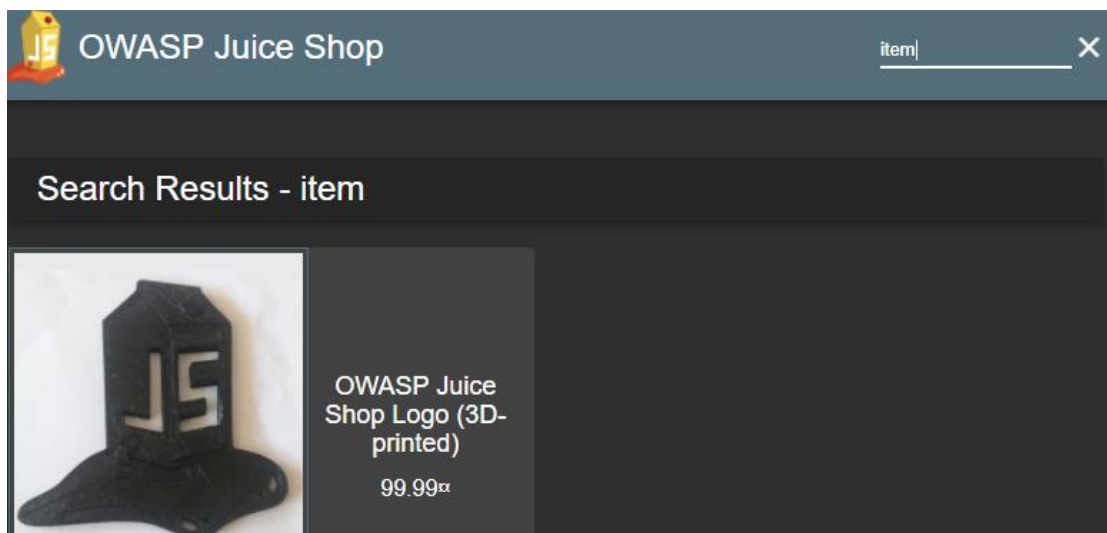


Рисунок 3.8 – Виведення параметрів пошуку

На рисунку 3.8 видно, що параметр пошуку, заданий у полі вводу справа зверху, виводиться на сторінку.

Спробуємо ввести простий JavaScript код до поля пошуку. Результат зображено на рисунку 3.9.

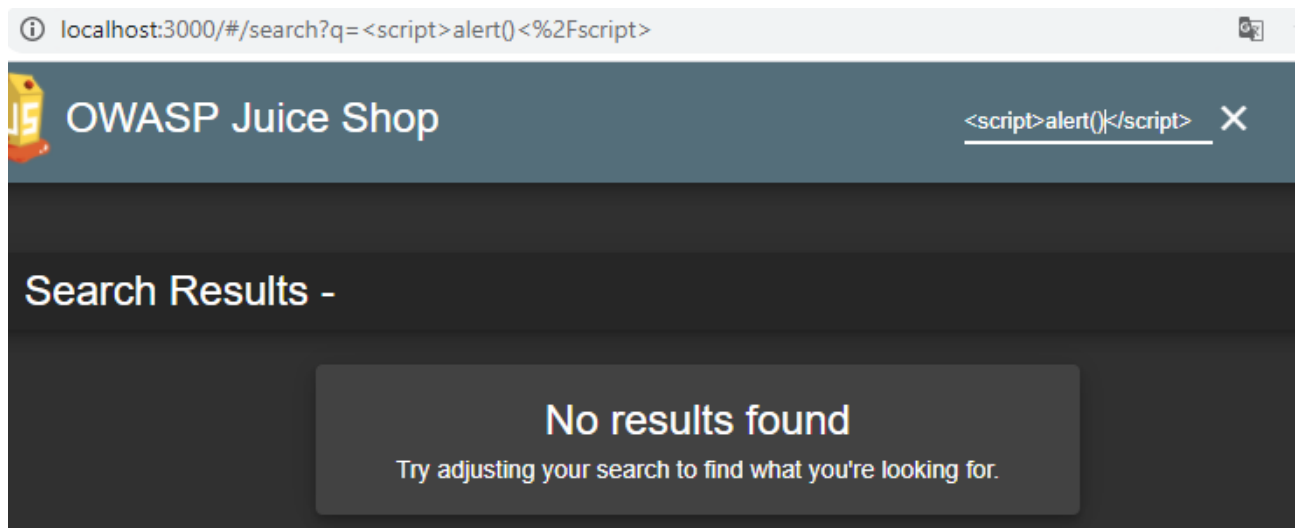


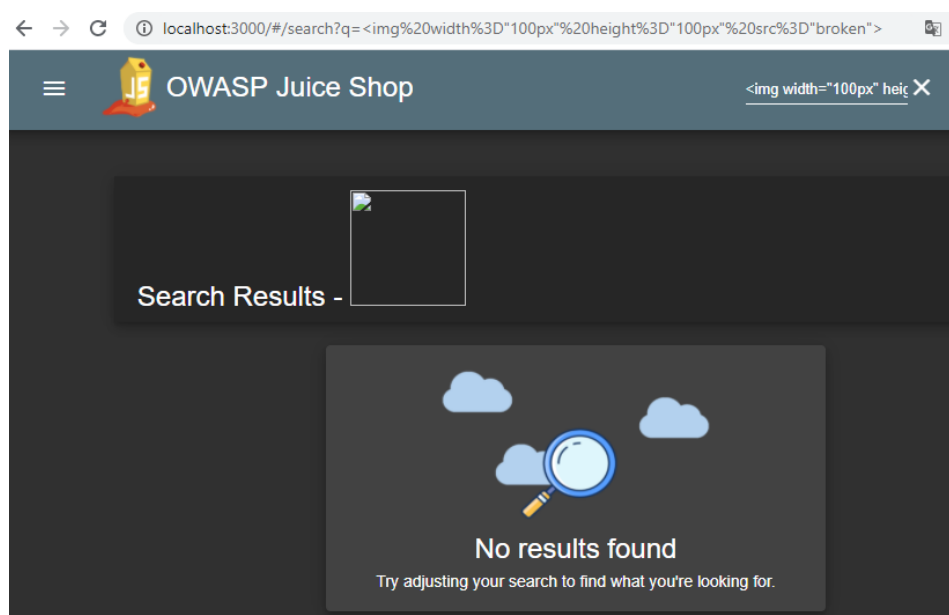
Рисунок 3.9 – Спроба пошуку з поганим вмістом

Після виконання запиту, бачимо, що «атака» не спрацювала – веб-сайт обробив запит та не показує його на сторінці. Інакше, з’явилося б модальне вікно alert. Це означає наявність захисту, але чи обробить більш складний запит? Також бачимо рядок пошуку, де наявне шкідливе наповнення – саме по такому посиланню повинна перейти жертва, щоб атака спрацювала.

Для виконання більш складної спроби атаки скористаємося Cheat-sheet від Burp Suite. У цьому документі наведено сотні різних варіантів виконання XSS ін’єкції, з різним маскуванням, для різних браузерів та їх версій, для різних елементів веб-сторінки та ще багато інших параметрів – таблиця показана на рисунку 3.10.

Через декілька спроб було з’ясовано, що додаток ігнорує тег script напрямую, але не оброблює інші HTML теги та взагалі виводить такі елементи на сторінку. На рисунку 3.11 можна побачити, що у результатах пошуку, додаток покаже картинку, після введення відповідного запиту.

Event:	Description:	Tag:	Code:	Copy:
onactivate				
Compatibility:    	Fires when the element is activated	custom tags ▾	<code><xss id=x tabindex=1 onactivate=alert(1)></xss></code>	 
onafterprint				
Compatibility:    	Fires after the page is printed	body ▾	<code><body onafterprint=alert(1)></code>	 
onafterscriptexecute				
Compatibility:    	Fires after script is executed	custom tags ▾	<code><xss onafterscriptexecute=alert(1)> <script>1</script></code>	 
onanimationcancel				
Compatibility:    	Fires when a CSS animation cancels	custom tags ▾	<code><style>@keyframes x{from {left:0;}to {left:1000px;}}:target {animation:10s ease-in-out 0s 1 x;}</style><xss id=x style="position:absolute;" onanimationcancel="alert(1)"></xss></code>	 



Така поведінка вже вказує на те, що вразливість існує, варто лише модифікувати запит.

Слід лише додати маленьку деталь та спостерігати за результатом.

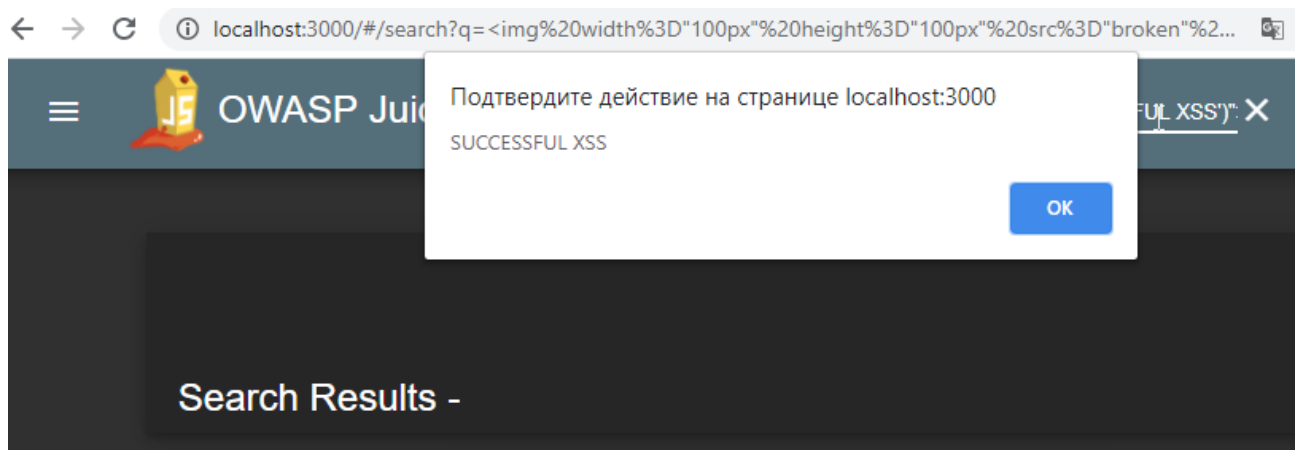


Рисунок 3.12 – Успішне використання вразливості

В оригінальному запиті було передано зображення (тег `<img>`), але шлях до зображення – неіснуючий. Якщо виникає така ситуація, то можна обробити таку помилку та зробити якусь дію, щоб зберегти гарний користувацький досвід, але такий функціонал не завжди використовується у добрих цілях. Так, навмисно вказавши поганий шлях до зображення, ми навмисно викликали функцію-обробник помилки, але замість обробки помилки, атакували користувача.

```
filterTable () {
  let queryParam: string = this.route.snapshot.queryParams.q
  if (queryParam) {
    queryParam = queryParam.trim()
    this.dataSource.filter = queryParam.toLowerCase()
    this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
    this.gridDataSource.subscribe((result: any) => {
      if (result.length === 0) {
        this.emptyState = true
      } else {
        this.emptyState = false
      }
    })
  } else {
    this.dataSource.filter = ''
    this.searchValue = undefined
    this.emptyState = false
  }
}
```

Рисунок 3.13 – Код функції, яка обробляє запит

Переглянувши код функції, яка перевіряє введені дані, на рисунку 3.13, можна побачити що тут насправді наявна частина, яка має перевіряти значення пошуку і ми вже стикалися з тим, що передавши тег script у відкритому вигляді нічого не вдасться, але слід було виконати JavaScript код інакшим шляхом і атака стає успішною.

Наведена вразливість – це DOM XSS, так як атака була виконана завдяки доданому на сторінку елементу, який в свою чергу виконав дію, а фронт-енд частина, в свою чергу, не впоралась з перевіркою контексту запиту. Перевірка існує, але не є досконалою.

У документації до додатку наведено список присутніх вразливостей. Отож, наступною вразливістю, яку варто згадати це Improper Input Validation, що означає «Неправильна перевірка вводу».

У попередніх розділах вже було сказано, що необхідно проводити жорстку валідацію даних, які вводить користувач. Особливо, коли нам відомо які саме дані користувач може або повинен відправити.

Гарним прикладом в додатку Juice Shop є форма відправки відгуку, де можна поставити оцінку від 1 до 5. Але що як хтось спробує відправити інший набір даних?

The image shows a 'Customer Feedback' form. It includes an 'Author' field with the value 'anonymous', a 'Comment' text area, a 'Rating' slider, a CAPTCHA section with the question 'What is 1+6+8?' and a 'Result' field containing '15'. A 'Submit' button is located at the bottom of the form.

Рисунок 3.14 – Вікно відправки відгуку

У наявному вікні наче і є ліміти, але це стандартний засіб, просто за допомогою HTML атрибутів. Правильність таких даних має бути також перевірена на серверній частині.

За допомогою Burp Suite перехопимо запит та подивимось його вміст на рисунку 3.15.

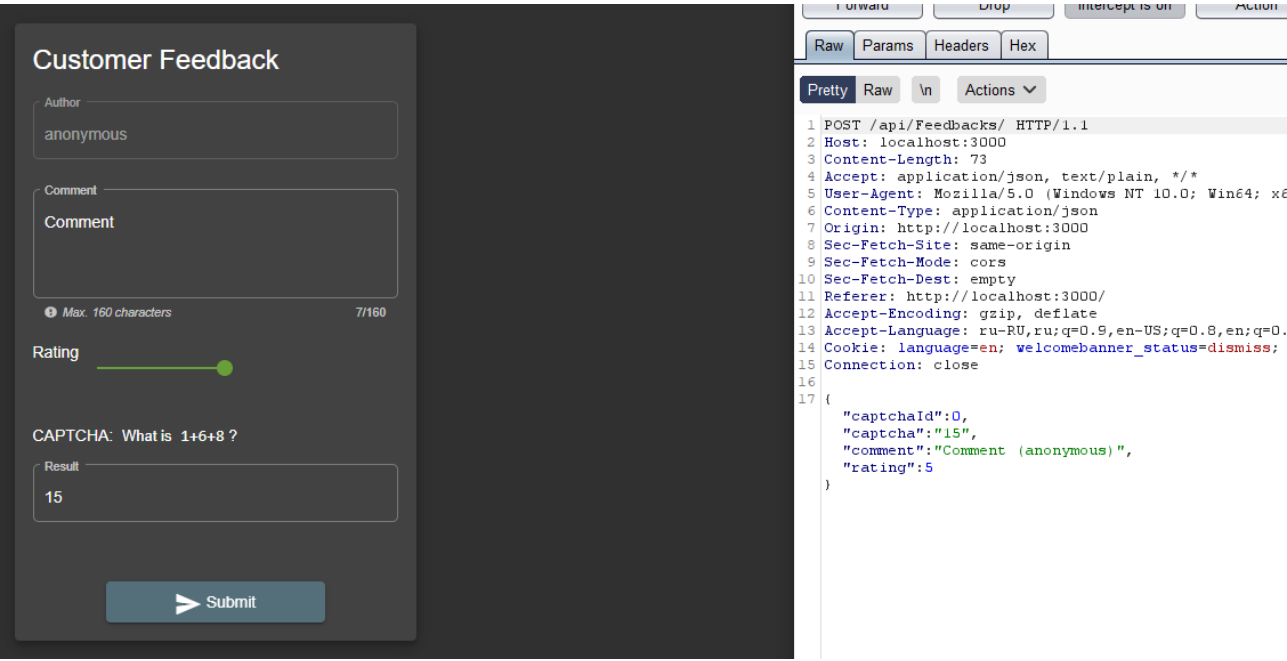


Рисунок 3.15 – Перехоплено запит на відправку відгуку

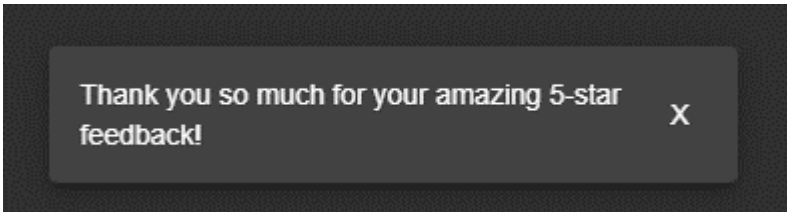


Рисунок 3.16 – Отримано відповідь від серверу

Отже, на наведених вище рисунках видно відправку запиту та результат його виконання. Спробуємо змінити відправлені дані – введемо значення показані на рисунку 3.17.

```
{
  "captchaId":0,
  "captcha":"15",
  "comment":"Comment (anonymous)",
  "rating":"lorem"
}
```

Рисунок 3.17 – Вводимо довільні дані

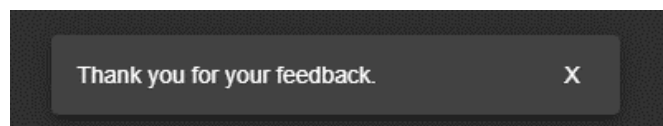


Рисунок 3.17 – Отримано відповідь від серверу

Бачимо, що сервер не повернув ніякої помилки, хоча замість оцінки, яка має бути цифровим значенням, було відправлено текст. У деяких випадках таке невірне значення може призвести до внутрішніх помилок та додаток перестане працювати. Також такі погані перевірки можуть стати причиною інших атак – SQL ін'єкцій або Stored XSS.

Усі перевірки які робляться на стороні користувача мають дублюватися і на стороні сервера, що є набагато важнішим, бо це напряду відповідає безпеці веб-додатку та усіх його даним.

На цьому веб-сайті можна зареєструватися. У модальному вікні для реєстрації є пункт «повторити пароль» - він присутній на багатьох сайтах та служить страхуванням від випадкового вводу символів. Маловірогідно, що користувач зможе випадково натиснути на якісь зайві клавіші два рази, та ще й щоб вони співпали.

Рисунок 3.18 – Вигляд вікна реєстрації

```

1 @Component({
2   selector: 'app-register',
3   templateUrl: './register.component.html',
4   styleUrls: ['./register.component.scss']
5 })
6 export class RegisterComponent implements OnInit {
7   public emailControl: FormControl = new FormControl('', [Validators.required, Validators.email])
8   public passwordControl: FormControl = new FormControl('', [Validators.required, Validators.minLength(5), Validators.maxLength(20)])
9   public repeatPasswordControl: FormControl = new FormControl('', [Validators.required, matchValidator(this.passwordControl)])
10  public securityQuestionControl: FormControl = new FormControl('', [Validators.required])
11  public securityAnswerControl: FormControl = new FormControl('', [Validators.required])
12  public securityQuestions!: SecurityQuestion[]
13  public selected?: number
14  public error: string | null = null
15

```

Рисунок 3.19 – Код функції реєстрації

Переглядаючи вихідний код функції, яка відповідає за це модальне вікно, видно, що валідація наявна, включаючи ту, яка нас цікавить.

Спробуємо зареєструватись, змінивши в запиті значення паролю, який ми вводимо для перевірки.

```

{
  "email": "email1@email.com",
  "password": "password",
  "passwordRepeat": "STRING",
  "securityQuestion": {
    "id": 13,
    "question": "Number of one of your customer or ID cards?",
    "createdAt": "2021-06-15T05:31:46.803Z",
    "updatedAt": "2021-06-15T05:31:46.803Z"
  },
  "securityAnswer": "12"
}

```

Рисунок 3.20 – Дані, які передаються на сервер при реєстрації

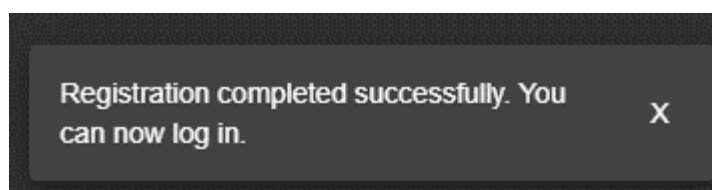


Рисунок 3.21 – Результат виконання запиту на реєстрацію

Відповідь, отримана від сервера, що зображена на рисунку 3.21, повідомляє нам, що на стороні сервера немає додаткової валідації введених даних – хоча б стосовно повтору паролю.

Перевіримо інші поля вводу на вразливість XSS. На сторінці користувача можна змінити юзернейм – введемо дані, які вже використовувались для викриття XSS вразливостей у минулих прикладах: ``.

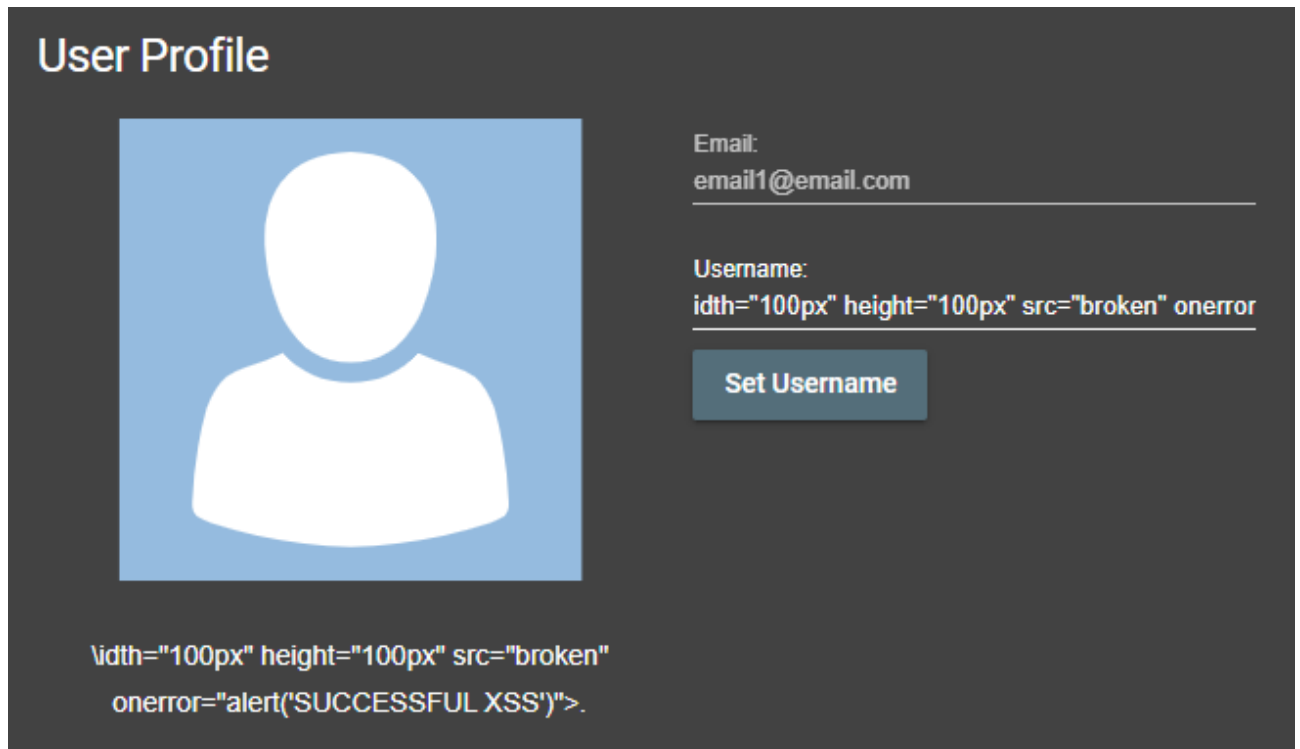


Рисунок 3.22 – Змінена інформація користувача

На рисунку 3.22 видно, що відбулась спроба видалення HTML тегу з тексту. Видалено таку частину: «` {
 if (result.length === 0) {
 this.emptyState = true
 } else {

```

Рисунок 3.30 – Частина коду, яка відповідає за «очищення» запиту

Усі техніки захисту мають підкріплюватися один одними. Поодинокого варіанту захисту може просто не вистачити, як, наприклад, у даному конкретному випадку. Слід було також врахувати захист на моменті виведення параметрів запиту на сторінку.

Наприклад, в чистому JavaScript це можливо за допомогою методу `textContent` – виведені дані будуть передані та сприйняті як текст і тоді можна не боятися виконання коду з подібних запитів. По ішому, це можна зробити за допомогою екранування. Замінивши спец символи юнікод-значеннями або провівши екранування кожного з цих символів – можна виводити текст і браузер не буде інтерпретувати його як код.

Існують дуже великі документи, які містять в собі способи імплементації тієї чи іншої вразливості. Дивлячись на такі величезні списки, стає зрозуміло, що неможливо захистити додаток від кожної з атак, особливо в ручну. Тому, існують автоматичні методи сканування на вразливості та, відповідно, бібліотеки, які збирають можливі способи захисту в одне місце. Слід використовувати обидва інструменти та виконувати тестування власного додатку час від часу, а бібліотеки мають бути вчасно оновлені.

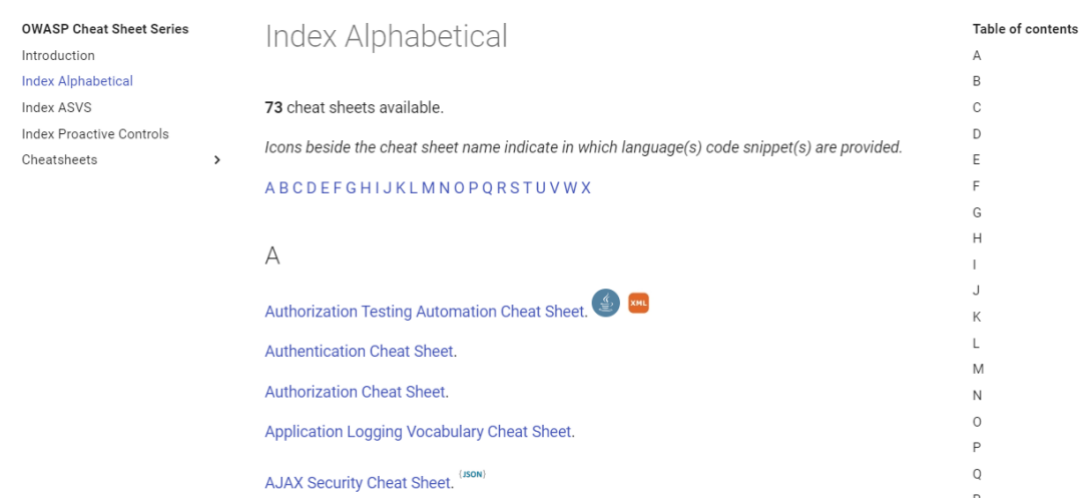


Рисунок 3.31 – Список Cheat Sheets від OWASP

Окрім вже готових бібліотек, існують подібні збірки вищеописаних документів, які містять всі зібрані дані про відомі вразливості: як їх використати та як від них захиститися. На рисунку 3.31 наведено лише зміст. Самих документів на даний момент наявно 73, а це лише один такий збірник.

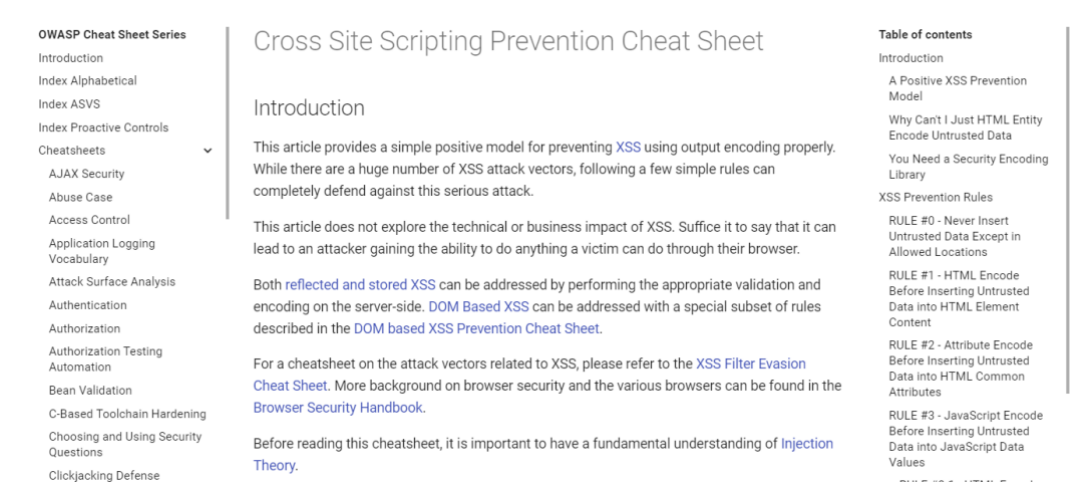


Рисунок 3.32 – Вигляд Cheat Sheet

Наступною знайденою вразливістю була непрацююча валідація даних, відправлена на сервер. Цю вразливість було знайдено при відправці відгуку.

У формі можна було поставити оцінку від 1 до 5, але перевірка довела, що ці значення були перевірені лише на стороні клієнта. Шляхом перехоплення запиту через Burp Suite, ми змогли передати змінене значення на сервер та отримали успішну відповідь. Відправлено було не тільки числове значення в невірному діапазоні, а й текстове значення, з яким також не виникло помилок чи проблем.

Це добре, що нічого не трапилось у даному випадку, але в іншій ситуації можна, з такими самими обставинами, програма може втрачати свою працездатність. Це в основному пов'язано з поганим програмним кодом, який неправильно оброблює помилки. При передачі іншого типу даних, від очікуваного, програма не зможе виконати відповідні операції та не буде знати що робити – в результаті перестане працювати.

Особливо вразливими є ті програми, які виводять контент від одних користувачів для інших. Приклад – чат. Якщо навіть програма буде достатньо стійкою та витримає обробку всієї неправильно введенної інформації, то ситуація на стороні клієнта може бути не такою доброю. Передавши погану інформацію, сервер розповсюдить повідомлення усім користувачам, і тоді зламатися може вже клієнтська частина веб-додатку – сайтом буде неможливо користуватися.

```
function changeDelay() {
 let delay = parseInt(delayElement.value);
 if (Number.isInteger(delay) && delay ≥ 0) {
 $.ajax({
 traditional: true,
 url: "/applyDelay",
 method: "POST",
 data: {
```

Рисунок 3.33 – Приклад жорсткої перевірки значень

На рисунку 3.33 наведено приклад жорсткої перевірки значень – дані в полі вводу мають бути лише цифровим значенням та лише більше або дорівнювати 0.

Інакше – запит не спрацює. Точно такий механізм повинен бути реалізований на серверній частині.

Наступною, однією з найбільш важливих вразливостей, було розглянуто SQL ін'єкцію при вході в особистий акаунт на веб-сайті.

Ця вразливість паралельно відкрила десятки нових джерел для можливих атак: новий функціонал, більше привілеїв, доступ до інформації інших користувачів, з іншими привілеями та інше.

Кроки першої важливості задля запобігання SQL ін'єкцій:

- використання підготовлених запитів (обов'язково із параметризацією)
- використання збережених процедур
- валідація вводу по спеціально підготованим спискам
- екранування всіх користувацьких даних

Додатковим засобом захисту може бути застосування найменших привілеїв.

Усі наведені вище пункти були детально розглянуті у попередньому розділі.

Дана атака є особливо небезпечною, бо за допомогою неї можна вільно спілкуватися із базою даних, яка може містити дуже велику кількість важливої та конфіденційної інформації. Важливо розуміти, що навіть якщо будь-яка з уразливостей була знайдена, але не явно експлуатована – це може свідчити про збір інформації та підготовку до більш масштабної атаки з використанням знайденої інформації на даному етапі.

## ВИСНОВКИ

В даній кваліфікаційній роботі було проаналізовано роботу веб-додатків та розглянуто основні вразливості та атаки, які можна здійснити.

Відповідні вразливості були визначені та обрані згідно до публікацій авторитетних організацій, які займаються збором усієї можливої інформації про безпеку веб-додатків задля подальшого її поширення у навчальних цілях.

Було наведено інформацію про десяток найбільш відомих вразливостей та атак, а згодом обрано декілька з них та розглянуто детальніше.

В другому розділі було детально описано варіанти можливих реалізацій відповідних атак та згодом, можливі варіанти запобігання цим самим атакам та поради щодо організації додатку та його захисту.

Було розглянуто конкретні запобіжні методи та рекомендації та наведено відповідні приклади.

В третьому розділі було змодельовано декілька попередньо розглянутих атак, а для цього було розгорнуто вразливий веб-додаток на локальній машині для демонстрації.

У ролі вразливого веб-додатку виступав веб-сайт, розроблений організацією OWASP – Juice Shop.

У процесі розгляду було знайдено та змодельовано такі види атак як: Reflected XSS, DOM XSS, SQL Injection, Broken Authentication, Improper Input Validation.

Знайдені вразливості були відповідно проаналізовані у наступних чи попередніх розділах та підрозділах. Наведено поради щодо запобігання появи чи використання таких вразливостей.

Наведений веб-додаток є повністю готовим макетом реального проекту, але є протилежністю найкращих практик та стандартів для веб-розробників: це інструмент для навчання, демонстрації та проведення практичних занять з вразливостями в сучасних веб-додатках. Тому, наведений веб-додаток не може бути використаний для реальних потреб.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Информационная безопасность открытых систем. Средства защиты в сетях / С. В. Запечников, Н. Г. Милославская, А. И. Толстой, Д. В. Ушаков. – Москва: Горячая линия-Телеком, 2008. – 558 с. – (Горячая линия-Телеком).
2. Основи захисту периметру корпоративних мереж [Електронний ресурс]. – 2013. – Режим доступу до ресурсу: <http://5fan.ru/wievjob.php?id=17306>
3. Розвивайте бізнес за допомогою ІТ-рішень Cisco [Електронний ресурс] – Режим доступу до ресурсу: [https://www.cisco.com/c/dam/global/uk\\_ua/solutions/small-business/pdf/Cisco\\_SMB\\_Buyers\\_Guide\\_final.pdf](https://www.cisco.com/c/dam/global/uk_ua/solutions/small-business/pdf/Cisco_SMB_Buyers_Guide_final.pdf)
4. Жуков Ю.В. Основы Web-хакинга: нападение и защита / Ю.В. Жуков. — СПб.: Питер, 2011, — 176 с.
5. Фролов А. В. Базы даних в Мережі інтернет: Практичний посібник по створенню Webдодатків з базами даних / А. В. Фролов., 2000. – 448 с.
6. Кузьменко Б. В. Захист інформації. Ч. 1. Організаційно-правові засоби забезпечення інформаційної безпеки / Б. В. Кузьменко, О. А. Чайковська. – К. : Вид. відділ КНУКІМ, 2009. – 83 с.
7. International Forum of Educational Technology & Society [Електронний ресурс]. – Режим доступу : <http://ifets.ieee.org/>.
8. List of Testing Tools. [Електронний ресурс] // guru99 – professional courses. – Режим доступу : <http://www.guru99.com/list-of-testingtools.html>
9. Девянин П. Н. Теоретические основы компьютерной безопасности [Текст]: учеб. пособие для вузов / П. Н. Девянин, О. О. Михальский, Д. И. Правиков, А. Ю. Щербаков. – М.: Радио и связь, 2000. – 192 с. : ил. ; 21 см.
10. Защита от несанкционированного доступа к информации. Термины и определения: утв. Гостехкомиссией России. – М.: Изд-во стандартов, 1992.
11. Куліш Д. Розробка захищеного веб-додатку ведення та обліку наукових заходів (конференцій) // Vydáno Publishing House «Education and Science» Praha. – 2016. – XII MEZINÁRODNÍ VĚDECKO – PRAKTICKÁ

KONFERENCE– [Електронний ресурс] – Режим доступу:  
[http://www.ukrnauka.ru/2016\\_NPRT/2016-05-23\\_A4\\_tom-9.pdf](http://www.ukrnauka.ru/2016_NPRT/2016-05-23_A4_tom-9.pdf)

12. Офіційний сайт компанії Positive Technologies. Дослідження вразливостей та загроз [Електронний ресурс] – Режим доступу:  
<http://www.ptsecurity.ru/research/threatscape/>

13. ПОНЯТТЯ, СТРУКТУРА ТА РІЗНОВИДИ ВЕБ-САЙТІВ [Електронний ресурс] – Режим доступу: <http://www.ndu.edu.ua/liceum/web.pdf>

14. Open Source Web Application Security Scanner [Електронний ресурс] – Режим доступу: <http://w3af.org/>

15. Методи тестування безпеки веб додатків [Електронний ресурс] – Режим доступу: <http://uastudent.com/osnovni-metody-testuvannja-bezpeky-vebdodatktiv/>

16. Knex.js - A SQL Query Builder for Javascript [Електронний ресурс] – Режим доступу: <http://knexjs.org/>

17. Knex cheatsheet – Devhints [Електронний ресурс] – Режим доступу: <https://devhints.io/knex>

18. Стаття «Призначення та функції веб - сервера» [Електронний ресурс]. – Режим доступу: <http://zametkinapolyah.ru/servera-i-protokoly/http-server-ili-veb-servernaznachenie-funkcii-i-rol-servera-v-http.html>

19. Threat and Vulnerability Management Standard [Електронний ресурс] – Режим доступу: <https://www.resolver.com/trust/policies/threat-vulnerability-management-standard>

20. Common Vulnerabilities and Exposures [Електронний ресурс] – Режим доступу: [https://en.wikipedia.org/wiki/Common\\_Vulnerabilities\\_and\\_Exposures](https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures)

21. The Open Web Application Security Project [Електронний ресурс] – Режим доступу: <https://owasp.org/>

22. Рейтинг мов програмування 2020 [Електронний ресурс] – Режим доступу: <https://dou.ua/lenta/articles/language-rating-jan-2020/>

23. Web Application Security Consortium [Електронний ресурс] – Режим доступу: <http://www.webappsec.org/>

|  |  |  |  |  |                    |    |
|--|--|--|--|--|--------------------|----|
|  |  |  |  |  | КІК.КБ.Б-125-06-ПЗ |    |
|  |  |  |  |  |                    |    |
|  |  |  |  |  |                    | 69 |

24. Common Weakness Enumeration [Электроний ресурс] – Режим доступу: <https://cwe.mitre.org/>

25. Burp Suite [Электроний ресурс] – Режим доступу: <https://portswigger.net/burp>

26. OWASP Juice Shop [Электроний ресурс] – Режим доступу: <https://github.com/bkimminich/juice-shop>

27. OWASP Automated Threats to Web Applications [Электроний ресурс] – Режим доступу: <https://owasp.org/www-project-automated-threats-to-web-applications/>

28. OWASP ZAP Applications [Электроний ресурс] – Режим доступу: <https://owasp.org/www-project-zap/>

29. OWASP Top Ten [Электроний ресурс] – Режим доступу: <https://owasp.org/www-project-top-ten>